LA-UR-14-28528

Approved for public release;
distribution is unlimited.

Title:	Experiments at Scale with In-Situ Visualization Using ParaView/Catalyst in RAGE
Author(s):	Robert J. Kares Applied Computational Physics (XCP) Division Los Alamos National Laboratory
Intended for:	Report October 31, 2014

Form 836 (7/06)

This page is intentionally left blank.

Experiments at Scale with In-Situ Visualization Using ParaView/Catalyst in RAGE

Robert J. Kares

Applied Computational Physics (XCP) Division, Los Alamos National Laboratory
Los Alamos, NM 87545

In this paper I describe some numerical experiments performed using the ParaView/Catalyst in-situ visualization infrastructure deployed in the Los Alamos RAGE radiation-hydrodynamics code to produce images from a running large scale 3D ICF simulation on the Cielo supercomputer at Los Alamos. The detailed procedures for the creation of the visualizations using ParaView/Catalyst are discussed and several images sequences from the ICF simulation problem produced with the in-situ method are presented. My impressions and conclusions concerning the use of the in-situ visualization method in RAGE are discussed.

I. Introduction

Recently, there has been much discussion in connection with future program plans for exascale computing about the need to generate visualizations directly from within a running physics application as a problem is being computed in order to avoid writing out large volumes of simulation data to disk. Such an approach to visualization and data analysis is commonly termed "in-situ" visualization. The argument goes that in the exascale era, the scaling of rotating media will not be able to keep pace with that of processors and networks, and so the disks will become a serious performance bottleneck and the traditional work flow method of post-processing simulation data written to disk in order to extract visualization and analysis results will have to be abandoned in favor of the in-situ method. Whatever one thinks of the merits of this argument, it is clearly of some interest to begin investigating the application of in-situ visualization to realistic simulation problems in order to develop a better understanding of the advantages and limitations of such an approach.

I have undertaken such an investigation using a particular in-situ visualization software package, the ParaView/Catalyst software [1], installed in the Los Alamos radiation-hydrocode RAGE [2]. As part of a 2013 Advance Strategic Computing (ASC) program Level II milestone [3], the Los Alamos ASC code RAGE was modified to allow it to render images directly from a running problem using the ParaView/Catalyst library. These modifications consisted of installing ParaView adaptor library calls in RAGE to copy data from the adaptive mesh refinement (AMR) data structures used by RAGE into a VTK unstructured grid format suitable for use by the ParaView/Catalyst package and linking the RAGE code with the ParaView/Catalyst library. The

adaptor library calls in RAGE convert the 3D AMR mesh used by RAGE into an unstructured 3D mesh that is used by the ParaView/Catalyst library to render images directly from the running RAGE problem under the control of a custom ParaView visualization pipeline defined by the user. The visualization pipeline which defines the image to be rendered must be created interactively by the user with the ParaView software in a separate off-line step and is exported to ParaView/Catalyst as a Python .py file containing the pipeline definition. This .py file then becomes a part of the input to RAGE that defines the images to be produced. In addition to providing the linkage to ParaView/Catalyst in RAGE, the Level II milestone development work also added some custom features to RAGE for in-situ visualization that can be used in combination with ParaView/Catalyst. One of these is the so-called automatic camera [4] developed by the LANL visualization research group that attempts to find and track regions of the problem of interest as they change over time. The purpose of the investigation described in this report was to exercise these new in-situ visualization capabilities of the RAGE code by running a state-of-the-art 3D RAGE simulation problem at scale on the ASC Cielo supercomputer producing images with in-situ visualization. I begin with a brief description of the simulation problem chosen for this investigation and how it was run using the RAGE code.

II. 3D Simulation of a ICF Implosion with RAGE

The problem chosen for this investigation was the 3D simulation of the implosion of an Inertial Confinement Fusion (ICF) capsule using a pressure drive that was not spherically symmetric. The ICF capsule in question is a hollow plastic sphere with an outer radius of 425 μm and an inner radius of 400 μm. The center of the sphere is filled with an equimolar mixture of deuterium/tritium gas at a density of $2.5\ \mathrm{mg/cm^3}$ which acts as the fuel for the ICF target. The implosion of the capsule is driven by an energy source in the outer layers of the plastic shell that simulates the effect of laser beams shining on the capsule depositing energy into these outer layers. This energy source is not spherically symmetric but instead has a polar variation in intensity of the form $P_\ell(\cos\theta)$ with $\ell = 30$. The purpose of this imposed variation is to simulate the effect of real angular variations in the intensity of the laser beams driving the capsule in a real ICF implosion experiment. Note, however, that the perturbation chosen for the capsule's pressure drive has an angular variation which is only in the polar direction. Hence, while the capsule's pressure drive is no longer spherically symmetric, it is still axisymmetric about the polar axis of the capsule. This simplification allows one to carry out the initial phase of the implosion as a 2D axisymmetric simulation and thereby achieve an enormous savings in computational resources. The overall purpose of this simulation was to explicitly demonstrate how turbulence arises in the fuel as result of asymmetries in the pressure drive on the capsule.

This simulation problem has already been extensively studied using 3D RAGE, and the results from these studies are described in some detail in the papers [5-7] of Thomas and Kares. The goal in this investigation was not break any new physics ground, but rather to attempt to use the

new in-situ visualization capabilities of RAGE to reproduce selected visualizations from these papers [5-7]. The original visualizations that appear in these papers [5-7] were produced using the conventional post-processing method with the parallel version of the EnSight visualization software, the EnSight Server Of Servers (SOS) [8]. These visual images were selected as a target product for the in-situ investigation because they represent realistically complex visual representations of the data that have been found to be useful for understanding the physics of the simulation, and that should be readily produced by the in-situ method.

The way in which the current simulation was run using RAGE is very similar to the computational procedure employed in the original ICF capsule simulations described in Reference 5. I began with a 2D axisymmetric RAGE simulation of the capsule implosion that ran from $t = 0$ out to a selected link time of $t = 1.4$ ns with a maximum AMR spatial resolution of 0.4 μm in 2D. At the chosen link time $t = 1.4$ ns, I rotated one quadrant of the 2D axisymmetric problem into 3D to create a 3D octant version of the axisymmetric data and continued this octant simulation to late time using fully 3D hydro in RAGE. The 3D RAGE simulation in the present investigation utilized the same AMR zoning strategy that is described in Reference 5 with one important difference. In the present investigation I forced uniform zoning at the highest level of AMR resolution over the entire capsule in 3D whereas in Reference 5 the highest resolution zoning was only utilized in the spatial region near the plastic/gas interface. The effect of this change was to create a larger computational problem that was more appropriate for the Cielo generation of machine. From $t = 1.4$ ns to $t = 1.5$ ns a uniform spatial zoning in 3D of 0.2 μm was utilized over the entire capsule. From $t = 1.5$ ns to $t = 1.6$ ns the spatial zoning in the capsule was increased to 0.1 μm. For simulation times later than $t = 1.6$ ns the simulation utilized 0.05 μm zoning over the entire capsule resulting in a 3D RAGE simulation running on 14,400 CPU's of Cielo with a total mesh size of 3.19 billion AMR cells.

III. In-Situ Visualizations of the 2D RAGE Simulation

The methodology of the RAGE simulation for the ICF capsule problem was ideal for aiding the process of learning about how to use ParaView/Catalyst for in-situ visualization. The initial 2D RAGE simulation problem was a very modest one, requiring only 800 CPUs on Cielo with about 983 K total cells in 2D and so provided an ideal test problem on which to perform initial experiments with the in-situ visualization infrastructure deployed in RAGE.

Before these experiments could commence, however, a considerable amount of background work needed to be done to build and install the ParaView software on the Cielo machine and the ViewMaster2 rendering cluster at LANL. Most of this work was done by Paul Weber of the ASC Production Viz team with help from John Patchett of the ASC Research Viz team. Paul built versions of the ParaView client and pvserver executables for both Cielo and ViewMaster2 and created module files for their use. Paul also built the adaptor library required to use ParaView/Catalyst with RAGE on Cielo and provided me with a complete build tree for RAGE

containing the in-situ visualization components from ParaView/Catalyst that could be used to modify and rebuild the RAGE code. Such a build tree is needed because the ICF capsule problem requires the installation of a custom energy source in the esrcs.f90 module of RAGE that needs to be compiled into the code to produce the asymmetric pressure drive used in the ICF problem.

My own modest contribution the this effort was to create some scripts for launching the client and pvserver components of ParaView to allow the user to run ParaView interactively in two different configurations: one in which both the client and pvservers were run on nodes of the ViewMaster2 cluster and one in which ParaView was run distributed with the pvservers on backend visualization nodes of the Cielo machine connected to a ParaView client running remotely on a desktop node of ViewMaster2. These scripts have been made available to users on Cielo and ViewMaster2.

Once the background work of building and deploying the infrastructure for running ParaView interactively and for using ParaView/Catalyst for in-situ visualization with RAGE was completed, the process of creating a visualization pipeline for use with the 2D RAGE simulation began. The first step in this process was to run the 2D RAGE simulation using ParaView/Catalyst to output RAGE mesh data at an appropriate timestep as an unstructured grid with corresponding scalar variables in the VTK unstructured grid format. This mesh data could then be read interactively into ParaView and used to create the visualization pipeline using ParaView.

To generate the RAGE data in VTK format, I added the following command lines to the RAGE input deck for the 2D ICF simulation:

```
!==============================================================================
! ----- PLOTS PARAVIEW IN SITU
!==============================================================================

do_pv_insitu = .true.                  ! if this is false next two must be also
do_pv_insitu_gate = .false.            ! allow gate filter on output
do_pv_insitu_camera = .false.          ! allow camera to move by data

pv_use_python = .true.                 ! python pipeline vs hardcoded pipeline
pv_python_script = 'write.py'          ! if python, execute this script

pv_insitu_dt = 0.0025e-9               ! time delta for coprocessing

npv_insitu_mesh = 9                    ! number of insitu variables and names
pv_insitu_mesh(1) = 'rho', 'grd', 'mat', 'prs', 'v02', 'vel', 'xdt', 'ydt', 'zdt'
```

These lines cause ParaView/Catalyst to execute the referenced Python file “write.py” which writes out the requested RAGE mesh data at the indicated time interval in VTK format as the simulation runs. The detailed contents of the “write.py” file are shown in Fig. 1. Running the

RAGE simulation with this input generates a sequence of numbered dump directories of the form:

grid_*cyclenumber*

together with a corresponding grid_*cyclenumber*.vtm file containing the domain decomposed RAGE data in VTK format. Each dump directory contains mesh and variable data files, a file per processor, in the form:

grid_*cyclenumber*_0_*processornumber*.vtu

Each .vtm file contains metadata describing the domain decomposed output for the dump data on the corresponding timestep and this sequence of .vtm files can be opened with ParaView and used to create a visualization pipeline to display the simulation output over time. Data dumps in this format were generated by running RAGE on 800 Cielo processors, and selected dumps were then copied over to ViewMaster2 and used to create the pipeline definition for the in-situ visualization.

The target images to be produced in-situ from the 2D RAGE simulation were designed to be as close as possible to the images in Fig. 1 of Reference 5 and represent a straight-forward, albeit non-trivial, visualization of the 2D simulation results for the ICF implosion problem. The RAGE data dumps copied over from Cielo to ViewMaster2 were opened by running both the ParaView client and pvserver processes on rendering nodes of ViewMaster2, and the visualization pipeline required to generate the target images was interactively created in ParaView. The resulting pipeline definition was then exported from ParaView as a Python file called "ICFpicmaker.py" in a form suitable for use with Catalyst by using the Coprocessor plugin in ParaView. This .py file was then copied over to Cielo and used as input to define the in-situ visualization to be generated by the 2D RAGE simulation. The contents of the "ICFpicmaker.py" file which defines the Catalyst visualization pipeline for the 2D RAGE simulation is shown in Fig. 2.

One challenge in creating images for the 2D simulation arises from the nature of the problem being simulated which is that of an implosion. Between $t = 0$ and $t = 1.75$ ns the radius of the imploding ICF capsule shrinks by a factor of 8. This means that a visualization with a fixed view transformation will show the capsule shrinking more and more as time progresses until much of the interesting detail of the evolution of the shocks and vorticity in the gas at the center of the capsule becomes increasingly less visible as time goes on. In order to deal with this problem, I used the automatic camera feature of the in-situ visualization capabilities in RAGE to track the center of the shrinking capsule.

The in-situ images for the 2D RAGE simulation were generated by adding the following command lines:

```
!=============================================================================
! ----- PLOTS PARAVIEW IN SITU
!=============================================================================

do_pv_insitu = .true.                     ! if this is false next two must be also
do_pv_insitu_gate = .false.               ! allow gate filter on output
do_pv_insitu_camera = .true.              ! allow camera to move by data

pv_use_python = .true.                    ! python pipeline vs hardcoded pipeline
pv_python_script = 'ICFpicmaker.py'       ! if python, execute this script

pv_insitu_dt = 0.0025e-9                  ! time delta for coprocessing

npv_insitu_mesh = 9                       ! number of insitu variables and names
pv_insitu_mesh(1) = 'rho', 'grd', 'mat', 'prs', 'v02', 'vel', 'xdt', 'ydt', 'zdt'

pv_insitu_camera(1) = 0,0,0,0,2,0,0,0,0   ! 0=NOAUTO, 1=ZOOMOUT, 2=ZOOMOUTIN

pv_insitu_xmn(1) = 9*-400e-4              ! initial camera bounds
pv_insitu_xmx(1) = 9*400e-4
pv_insitu_ymn(1) = 9*-400e-4
pv_insitu_ymx(1) = 9*400e-4

pv_insitu_camera_weight(1) = 9*1          ! number of weights used in auto-camera
pv_insitu_camera_previous_bounds(1) = 9*1 ! number of previous bounds to use
pv_insitu_camera_max_frames(1)= 9*1       ! maximum of frames before zooming-in
pv_insitu_camera_x_bins(1)=9*200          ! number of spatial bins in x dimension
pv_insitu_camera_y_bins(1)=9*400          ! number of spatial bins in y dimension
pv_insitu_camera_z_bins(1)=9*1            ! number of spatial bins in z dimension
pv_insitu_camera_s_bins(1)=9*100
```

to the 2D RAGE input deck. The “ICFpicmaker.py” file referenced in the input deck contains the Catalyst pipeline definition. In the above input command lines I have turned on the automatic camera and configured it to zoom in or out based upon the time variation in the RAGE variable v02 which is the volume fraction of the gas at the center of the capsule and has a value that ranges from 0 (no gas in a cell) to 1 (the cell is pure gas). To help the automatic camera to correctly initialize, in the above input I have set the initial camera bounds variables pv_insitu_xmn/xmx and pv_insitu_ymn/ymx to 400 μm, in order to frame the initial dimensions of the capsule. The pv_insitu_camera variables were set by a process of trial and error in order to achieve the smoothest zooming possible. In particular, the x_bins, y_bins, z_bins and s_bins variables were all chosen to spatially sample the v02 variable over the area of the entire problem with enough bins to usefully track the relatively small area at the center of the full problem that contains the shrinking capsule center. By configuring the automatic camera to zoom in order to follow the time evolution of the shrinking gas region, I was able to keep the size of the capsule image roughly constant during the implosion.

In order to keep the capsule centered in the vertical direction of the image, hand coding was added to the WriteXRAGEImages routine at the end of the ICFpicmaker.py file in Fig. 2(g) in the section labeled “Automatic camera bounds”. This hand coding fetches the value of xmax, the

maximum dimension of the camera field of view in the x direction (which is also the vertical direction in the image) computed by the automatic camera, and then resets the camera bounds in the x direction to force them to be -xmax to xmax, thus centering the capsule in the vertical direction of the resulting image. This technique, first suggested to me by Boonth Nouanesengsy of the ASC Research Viz team, works well in keeping the capsule centered in the image as the automatic camera zooms.

Note that hand coding has also been added at the end of the class Pipeline definition in the "ICFpicmaker.py" file in Figs. 2(e) and 2(f) to add a time annotation and information labels to the final rendered images. These annotations were not exported using the Coprocessor plugin in ParaView but were added later by hand in order to document the final rendered images produced.

Fig. 3 shows the final complete time sequence of images generated using the in-situ visualization infrastructure in RAGE with the above command inputs for the 2D RAGE ICF implosion simulation. Each image is a time snapshot from the 2D RAGE simulation. In the lower portion of each snapshot the plastic shell is colored by pressure while the gas is colored by the gradient of pressure. In the upper portion of each snapshot the plastic shell is colored by density while the gas is colored by the azimuthal component of the gas vorticity. The white contour represents the position of the interface between the plastic and the gas. These still images compare well with the target images in Fig. 1 of Reference 5 that I set out to produce. So the results of this aspect of the investigation were quite satisfactory.

However, the animation of the full sequence of in-situ produced images does not have the smooth camera zoom that one would really want for a presentation quality animation product. Even with the above relatively fine sampling parameters for the camera x and y bins variables, the automatic camera still zooms in a stepwise fashion. I believe that the origin of this problem has to do with the fact that the 2D RAGE simulation is set up so that the capsule region is embedded in a much larger, coarsely zoned mesh that surrounds the capsule and whose function is to insure that the capsule remains far from the problem boundaries throughout the duration of the simulation. This type of set up is how problems are typically run in RAGE and it means that the region where most of the action is in the simulation has relatively small spatial dimensions compared with that of the total problem. In this case, for example, the initial size of the capsule is roughly 425 μm while the full mesh runs from 0 to 3500 μm in the x direction and from -3500 to 3500 μm in the y direction. Thus, the linear dimension of the region of the capsule where the gas volume fraction variable v02 is used to control the camera zoom is initially only about 425/3500 = 12% of the total linear dimension in the x direction and this region only becomes smaller as the implosion progresses. In its current version the automatic camera uses the pv_insitu_camera_x_bins and pv_insitu_camera_y_bins to divide the entire problem region into uniform sized bins and uses the sampling in these bins to control camera zoom. In this case even with the grid of 200 X 400 bins used in the current problem, I believe that there were not enough bins in the spatial region of interest to result in smooth motion for the automatic camera. I would

like to suggest that some additional inputs be added the automatic camera to allow the user to define the spatial bounds within which the sampling for the camera is to take place rather than having these bounds always be the full dimensions of the problem. This might significantly improve the smoothness of the camera motion for the ICF implosion problem and other RAGE simulations of interest.

IV. In-Situ Visualizations of the 3D RAGE Simulation

In the 2D phase of the RAGE simulation discussed above, the visualization problem was straightforward. Here we wanted to follow the motion of the shocks in the gas and the associated development of the gas vortices in 2D and the images produced in-situ and displayed in Fig. 3 are well suited for that purpose. The visualization problem in the 3D phase of the simulation is considerably more challenging. The overall goal of the simulation is to exhibit the mechanism by which the asymmetric pressure drive on the capsule leads to three-dimensional turbulence in the gas and this involves both a very large scale computational problem and a difficult visualization challenge. The visualization challenge is to render a representation of the gas vortices in 3D that shows how the fully 3D interaction of initially well-ordered, axisymmetric vortex rings devolves into the complex tangle of worm vortices that characterize a fully turbulent state for the gas. What is needed is a three-dimensional representation of the vorticity field inside the gas that clearly exhibits the vortex cores and their dynamical evolution over time. Experience has shown that a good approach to this problem is to volume-render the total vorticity in the gas with a transfer function that has been chosen to emphasize the high vorticity regions of the vortex cores. Fig. 2 of reference 5 shows some time snapshots from such a volume-rendered representation of the vorticity field in the ICF capsule produced using the conventional post-processing method with the parallel version of EnSight. Ideally, in the current investigation I wanted to generate a similar sequence of images in-situ using the ParaView/Catalyst infrastructure in RAGE.

Unfortunately, I was not able to successfully achieve this goal with the current 4.1.0 version of ParaView. ParaView is able to volume-render a scalar field on a structured grid that is distributed over multiple processors. It cannot volume-render directly from an unstructured grid. However, the adaptor library interface currently installed in RAGE produces an unstructured grid representation of the AMR mesh to present to ParaView/Catalyst and there does not appear to be a method currently in ParaView which allows the user to resample the unstructured grid data onto a structured grid with specified physical and logical dimensions to produce a structured representation of the data distributed across multiple processors that is suitable for input to the ParaView volume-renderer. There is an existing filter in ParaView called Resample With Dataset that is supposed to allow the user to resample unstructured data onto a structured grid but there are several problems with using this filter for the current investigation. First, there does not appear to be any straightforward way of specifying a structured box at a defined spatial location with specified physical and logical dimensions to use as the target for the resampling operation.

And second, the result of this resampling process is a structured dataset that lives on processor 0 which makes the operation not scalable to large datasets. After considerable back and forth email with Patrick O'Leary and Andy Bauer of Kitware about this problem, I was forced to conclude that I could not use volume-rendering to represent the vorticity field in the current investigation.

As a result I was forced to fall back on an older visualization technique I have used in the past for this problem, namely, to represent the vortex tubes in 3D by an isosurface of constant total vorticity at some appropriately chosen value. While this method is by no means ideal, it does produce a visual representation of the vorticity field that can usefully demonstrate the process by which unstable azimuthal wave growth on the interacting vortex rings leads to the fully turbulent development of the gas. One difficulty with this technique, however, is that during the final compression of the capsule in the 3D simulation after t = 1.4 ns, the peak total vorticity increases by about an order of magnitude and this makes it is essentially impossible to pick a single value for the vorticity isosurface that provides a useful visualization of the vortex tubes over a long period of simulation time. The net result is that as the simulation progresses it is necessary to periodically adjust the value of the vorticity isosurface chosen in order to track the evolution of the vortex tubes over time.

Beginning with a restart dump from the 2D simulation at t = 1.4 ns, I used 3D RAGE to rotate one quadrant of the 2D axisymmetric problem into 3D by performing a 90 degree rotation about the symmetry axis of the capsule to create a 3D octant version of the axisymmetric data and then used this data to initialize the 3D RAGE simulation. To create a visualization pipeline appropriate for approximating the images from Fig. 2 of Reference 5 using the isosurface rather than the volume-rendered representation for the vorticity field in the gas, I began by running 3D RAGE with the "write.py" script for a few cycles after the link time t = 1.4 ns in order to dump out some mesh data in the VTK unstructured data format to use as input to the pipeline construction process. As in the 2D case, I copied the VTK data over to ViewMaster2 and used ParaView to interactively construct the pipeline. I then used the Coprocessor plugin to export this pipeline as a Python file called "VortTubes.py" and edited this file to add time annotations and labels for the images. The result was the "VortTubes.py" script shown in Fig. 4.

Adding the following command lines for ParaView/Catalyst to the RAGE input desk:

```
!================================================================================
! ----- PLOTS PARAVIEW IN SITU
!================================================================================

do_pv_insitu = .true.                    ! if this is false next two must be also
do_pv_insitu_gate = .false.              ! allow gate filter on output
do_pv_insitu_camera = .false.            ! allow camera to move by data

pv_use_python = .true.                   ! python pipeline vs hardcoded pipeline
pv_python_script = 'VortTubes.py'        ! if python, execute this script

pv_insitu_dt = 0.0005e-9                 ! time delta for coprocessing
```

```
npv_insitu_mesh = 9                        ! number of insitu variables and names
pv_insitu_mesh(1) = 'rho', 'grd', 'mat', 'prs', 'v02', 'vel', 'xdt', 'ydt', 'zdt'

pv_insitu_camera(1) = 0,0,0,0,2,0,0,0,0   ! 0=NOAUTO, 1=ZOOMOUT, 2=ZOOMOUTIN

pv_insitu_xmn(1) = 9*-400e-4               ! initial camera bounds
pv_insitu_xmx(1) = 9*400e-4
pv_insitu_ymn(1) = 9*-400e-4
pv_insitu_ymx(1) = 9*400e-4

pv_insitu_camera_weight(1) = 9*1           ! number of weights used in auto-camera
pv_insitu_camera_previous_bounds(1) = 9*1 ! number of previous bounds to use
pv_insitu_camera_max_frames(1)= 9*1        ! maximum of frames before zooming-in
pv_insitu_camera_x_bins(1)=9*200           ! number of spatial bins in x dimension
pv_insitu_camera_y_bins(1)=9*400           ! number of spatial bins in y dimension
pv_insitu_camera_z_bins(1)=9*1             ! number of spatial bins in z dimension
pv_insitu_camera_s_bins(1)=9*100
```

I then ran the 3D RAGE simulation forward in time from the link at t = 1.4 ns using the AMR zoning strategy described in Section II above. Fig. 5 shows a sequence of time snapshots from this 3D simulation created using the ParaView/Catalyst based in-situ visualization capability in the RAGE code together with the "VortTubes.py" pipeline definition of Fig. 4. In each of the snapshots the vertical face of the gas is colored by the azimuthal component of vorticity and the horizontal face is colored by the gradient of pressure. The off-white surface is the plastic/gas interface. The gray tubes visible in the snapshots are isosurfaces of constant total vorticity and represent counter-rotating vortex rings in the gas. From t = 1.4 ns to t = 1.5 ns the isosurface value chosen for the total vorticity was $5\times10^{10}\,\mathrm{sec}^{-1}$ and for the period from t = 1.5 ns to t = 1.6225 ns this value was increased to $2.5\times10^{11}\,\mathrm{sec}^{-1}$. These vortex tubes are colored by the axial component of vorticity whose color palette has been chosen so that the zero level is represented in gray. At link time t = 1.4 ns the 3D simulation begins from a completely axisymmetric state with the vorticity purely in the azimuthal direction. As the 3D simulation progresses, the appearance of alternating yellow and blue regions on the gray tubes indicate the development of a non-zero axial component for the total vorticity as a result of the growth of 3D instabilities. The images in Fig. 5 were all rendered with the same fixed view transformation so the shrinking size of the imploding capsule is readily apparent.

An examination of the gas bubbles nearest the polar axis of the capsule in Fig. 5 shows that the vortex rings in that region are undergoing some type of turbulent evolution as the appearance of the blue and yellow regions on the rings indicate. However, because of the choice of isosurface value, most of the very intricate structures in the evolving vortex tubes in this region are completely invisible in this sequence of images. To exhibit these structures with in-situ visualization I had to backup and restart the 3D simulation from t = 1.55 ns with the rendered view zoomed in on the capsule and with a value for the vorticity isosurface of $1\times10^{12}\,\mathrm{sec}^{-1}$ more appropriate for observing the dynamical evolution of the vortex cores in this region of the gas. Fig. 6 shows the resulting sequence of time snapshots of the evolving vortex tubes. This

sequence clearly illustrates what can be missed in this problem without a judicious choice for the vorticity isosurface value. I happened to already know that 1×10^{12} sec^{-1} is a good choice for the isosurface in this case because I had previously done a great deal of interactive data exploration on this problem. Without this a priori knowledge of the correct value to pick, I might have had to do a lot of expensive and time consuming recomputing of this problem in order to obtain a useful understanding of just how the vortex tubes evolve to a fully turbulent state, which is the entire point of doing the simulation in the first place. One topic that needs to be highlighted in any discussion of in-situ visualization is the cost of recomputing portions of the problem because of an incomplete a priori knowledge of what parameters to pick in order to effectively observe with in-situ visualization the phenomena that the simulation was designed to study in the first place. The sequence in Fig. 6 represents an extra 3.5 days of computing on 14,400 Cielo CPUs or roughly 1.2 million CPU-hours of computing required to recompute the in-situ generated images with a more appropriate choice for the vorticity isosurface. The costs of not having available the a priori knowledge that comes from interactive data exploration can be non-trivial as this example illustrates. One possible method to avoid this problem is to render numerous images of the simulation with multiple views and multiple values for the vorticity isosurface. Unfortunately, I was unable to experiment with rendering multiple images on the same timestep with the in-situ infrastructure in RAGE within the context of the current investigation. This is an approach that needs to be further investigated in the future.

V. Resource Usage by Catalyst in RAGE

It is of some interest to examine the cost in run time and memory usage of generating images in-situ with the Catalyst-based visualization infrastructure deployed in RAGE. To get some idea of the magnitude of these costs I performed the following simple test using my 14,400 CPU, 3.19 billion cell RAGE 3D ICF simulation. I selected a short 171 cycle long time segment from this simulation beginning at cycle 9545 corresponding to a simulation time of t = 1.632979 ns and ran this segment of the simulation forward in time both with and without the generation of in-situ images in RAGE starting from the same 1 TB restart dump. This time segment of the full simulation was chosen because during this period I forced uniform 0.05 μm zoning within a fixed spatial radius of 85 μm which encompasses the entire capsule so that while the AMR mesh adaptation is active during this period, the total AMR cell count remains essentially constant over time because the active region of the problem is already zoned at the maximum permitted AMR resolution. The 171 cycle length for the test segment was chosen to include a total of four calls to the in-situ visualization package. Both run time and memory use were measured using diagnostics obtained from the RAGE log file.

For measuring run time RAGE prints upon completion of a segment of the simulation the wallclock time for the execution of that segment. For the test run over the chosen time segment with the in-situ visualization package turned off, the observed wallclock time reported by RAGE

was 10964.2 secs. In contrast, for the test run over the chosen time segment with the in-situ visualization package turned on, the observed wallclock time was increased to 13566.2 secs. Thus, it appears that the use of the in-situ visualization package increased the total run time for the test run by 13566.2/10964.2 = 1.237 or roughly 23.7 %.

For measuring memory usage RAGE periodically prints to its log file a quantity called RSS_MAX which is the memory high water mark for the Resident Set Size, the total size of the process residing in memory, obtained from a call to the operating system. (Note that RSS_MAX is reset to zero at every restart.) This value is computed on each node of the job, and then the Min, Mean and Max for RSS_MAX over all the nodes is reported to the RAGE log file as a percentage of the total available memory on a node. For the test run over the chosen time segment with the in-situ visualization package turned off, the observed Mean value of RSS_MAX over the nodes was 31.69 % of the total available memory on a node. In contrast for the test run over the chosen time segment with the in-situ visualization package turned on, the observed Mean value of RSS_MAX increases to 45.16 % of the total available memory on a node. Thus, it appears that the use of the in-situ package increases the total RAGE memory usage by 45.16/31.69 = 1.425 or roughly 42.5 %.

In summary, while the total overhead cost to the simulation of using in-situ visualization observed in this simple test is about 24 % in terms of run time, the memory overhead cost is more serious. In the test case described above, turning on in-situ visualization resulted in a 42.5 % increase in memory use on a node compared to the case without calls to the in-situ package.

VI. Conclusions

The results of the experiments described in this paper clearly demonstrate that the ParaView/Catalyst-based visualization infrastructure combined with the automatic camera capability now deployed in RAGE can be used to produce some fairly sophisticated, useful visualizations in-situ from a running RAGE problem at scale on the Cielo machine. With ParaView I was able to interactively set up some relatively complex pipelines for visualizing my ICF simulation data and export these pipelines to Catalyst in order to render images directly out of the running RAGE code. Some of the new capabilities and challenges presented by this in-situ approach are readily apparent in the experiments described above.

One useful capability of such an approach is the ability to generate time snapshots of the data at a much higher frequency than is practical by writing the data out to disk and post processing it. This capability proved quite useful in the current investigation when generating time snapshots of the vortex tube evolution in 3D. This evolution is quite rapid and generating a large number of animation frames proved very interesting in observing the details of the tube interactions.

Monitoring the progress of the running simulation by generating time snapshots directly out of the RAGE code is another useful capability provided by the in-situ visualization infrastructure in RAGE. In the current investigation I found that having the RAGE code generating non-trivial visualizations as the problem ran was very useful for conveying a sense of the state of the problem, and for monitoring the progress of the calculation.

The in-situ approach presents some significant challenges as well. For example, long experience here at LANL with the visualization of complex 3D problems shows that looking at 2D slices through a 3D dataset quickly becomes inadequate for a comprehensive understanding of the problem. This suggests to me that the approach of adding parameter-controlled generic clips and isosurfaces to the input deck of a code, as Sandia is attempting to do with its Sierra framework, will prove totally inadequate for the purpose of using in-situ visualization for the useful analysis of real 3D problems. From a practical point of view, sophisticated visualization pipelines are required for understanding complex 3D problems, and it takes a sophisticated visualization tool user to create such a visualization pipeline appropriate for understanding a complex 3D simulation. This is a cost that will continue to have to be paid if science is to be successfully done with large scale computing.

One challenge that I continually observed during the course of this investigation was the level of effort that went into building and maintaining both ParaView and the ParaView/Catalyst infrastructure for RAGE. Building and maintaining a physics code linked to Catalyst is a rather complicated and time consuming task which involves considerable coordination with the code development team. The difficulty of this task should not be understated.

And as I pointed out in Section V above, the in-situ capability to produce images directly out of RAGE comes with some associated run time costs. In the simple test I described above I observed a roughly 24 % increase in run time and a 42 % increase in memory use as a result of turning on the production of in-situ images in the RAGE simulation. This result needs to be more thoroughly investigated and better understood. It seems likely that significant improvements in the memory usage of the in-situ visualization package are possible both by improvements to the adaptor interface to Catalyst in RAGE that avoid the deep memory copy currently being used by the interface and by improvements to the memory utilization of the user-defined rendering pipelines. In these tests I did not have sufficient knowledge of the underlying ParaView filter code to make any attempt at pipeline optimizations. A more sophisticated user might have better luck in constructing pipelines that minimize memory use in Catalyst.

Finally, as I have pointed out above, not having the a priori knowledge of how to choose parameters for the visualization like view transformations, palette ranges, and isosurface values can lead to costly recomputing of portions of a problem. In section IV above we saw that a less than optimal choice for the value of the vorticity isosurface resulted in the production of a visualization of the vortex tubes in which the dynamics of the tube interactions, the principal

phenomenon we wished to exhibit with the simulation, was essentially invisible. Correcting that error required significant recomputing to back up and render time snapshots with a value of the vorticity isosurface more appropriate for capturing the central phenomenon of interest. In fact, the problem of correctly choosing parameters for the in-situ visualization is perhaps the most challenging aspect of the in-situ approach. Clearly, combining in-situ visualization with a priori information obtained from interactive data exploration with the more conventional post-processing methodology makes the in-situ approach vastly more useful. Conversely, not having this information available can prove costly indeed. Proposals have been made to circumvent this problem by rendering multiple views with a range of parameter values, but I have not yet been able to test the utility of such an approach in the current investigation. It seems clear that in the absence of a priori knowledge that comes from interactive data exploration in a post-processing methodology, the ability to render multiple views with a range of parameters and to organize the resulting database of imagery is a minimal requirement for the successful application of in-situ visualization for scientific discovery with large scale computing. A specific framework for managing the production and viewing of large collections of images rendered from a running physics code using Catalyst, the so-called Cinema framework, is currently being developed [9], and it would be of some interest to experiment with this Cinema framework in the context of the 3D ICF simulation discussed here to examine its potential for mitigating some of the problems encountered in the current investigation.

VII. Acknowledgements

My sincere thanks to John Patchett of the LANL Research Visualization team for many interesting and useful discussions about the use of the in-situ visualization capabilities in RAGE. I would like to thank Paul Weber and John Patchett of LANL for building the ParaView/Catalyst software for the Cielo and ViewMaster machines, and for providing me with a working build environment for RAGE that included the Catalyst-based in-situ visualization capabilities. Thanks also to Boonth Nouanesengsy of LANL for his useful suggestions about how to use the automatic camera feature of RAGE. Finally, my thanks to Patrick O'Leary and Andy Bauer of Kitware, Inc. for answering some of my questions about the ParaView software.

VIII. References

1. A. C. Bauer, B. Geveci, and W. Schroeder, "The ParaView Catalyst User's Guide v1.0", Kitware, Inc. (2013). http://www.paraview.org/Wiki/images/4/48/CatalystUsersGuide.pdf

2. M. Gittings, R. Weaver, M. Clover, T. Betlach, N. Byrne, R. Coker, E. Dendy, R. Hueckstaedt, K. New, W. R. Oakes, D. Ranta, and R. Stafan, "The RAGE radiation-hydrodynamic code", *Comput. Sci. Disc.* **1**, 015005 (2008).

3. J. Patchett, J. Ahrens, B. Nouanesengsy, P. Fasel, P. O'Leary, C. Sewell, J. Woodring, C. Mitchell, L. Lo, K. Myers, J. Wendelberger, C. Canada, M. Daniels, H. Abhold and G. Rockefeller, "Case Study of In-Situ Data Analysis in ASC Integrated Codes", Los Alamos National Laboratory report LA-UR-13-26599 (2013). http://datascience.lanl.gov/data/papers/2013-2.pdf

4. J. Patchett and B. Nouanesengsy, private communication (2014).

5. V. A. Thomas and R. J. Kares, "Drive Asymmetry and the Origin of Turbulence in an ICF Implosion", Phys. Rev. Lett. **109**, 075044 (2012). http://arxiv.org/ftp/arxiv/papers/1210/1210.3364.pdf

6. V. A. Thomas and R. J. Kares, "Drive Asymmetry, Convergence and the Origin of Turbulence in ICF Implosions", Los Alamos National Laboratory report LA-UR-13-29528 (2013).

7. V. A. Thomas and R. J. Kares, "Drive Asymmetry, Turbulence and Ignition Failure in High Convergence ICF Implosions", Los Alamos National Laboartory report LA-UR-13-23446 (2013). http://permalink.lanl.gov/object/tr?what=info:lanl-repo/lareport/LA-UR-13-23446

8. "EnSight User Manual for Version 10.0", Computational Engineering International, Inc. (2014). http://www3.ensight.com/EnSight10_Docs/UserManual.pdf

9. J. Ahrens, S. Jourdain, P. O'Leary, J. Patchett, D. Rogers and M. Peterson, "An Image-based Approach to Extreme Scale In Situ Visualization and Analysis", paper submitted to Super Computing 2014 (2014). http://datascience.lanl.gov/data/papers/SC14.pdf

Appendix A. Problems Encountered with the ParaView/Catalyst Software

A few problems were encountered with the ParaView and Catalyst software during the course of the investigation that are worth documenting here. They are listed below under several broad categories.

Problems with the Coprocessor Plugin

I noted several problems when trying to use the Coprocessor plugin in ParaView to export a created pipeline to Catalyst:

1. When you export a pipeline to Catalyst, DO NOT give the created the Python script the name "test.py". As it turns out this is a special name, and it does not work with Catalyst.

2. If you create a color palette for a variable using fixed RGB color/data control points, and export the pipeline with the Coprocessor plugin, the corresponding GetLookupTableForArray call generated by the Coprocessor plugin fails to include the argument ColorSpace='RGB'. The result is that when the variable is rendered by Catalyst, a rather weird looking color map results because the default value of the argument is ColorSpace='HSV'and if the argument is not explicitly specified, Catalyst tries to interpret the RGB control points as HSV values instead. The argument ColorSpace='RGB' has to be added to the call by hand to correct this problem.

3. It is annoying that data values in the exported color palette definitions generated by the Coprocessor plugin are often represented in fixed rather than e format. For example, I often set palette min/max values to numbers like 2e12 and ended up with a fixed value in the form 2000000000000 in the exported script. This makes it hard to search the resulting script for values when you wish to edit them.

4. Naïvely trying to export text annotations created in ParaView using the Coprocessor plugin results in a pipeline that when used with Catalyst will fail to render an image.

5. If you create a pipeline with two different isosurfaces defined in it, and export this pipeline to Catalyst with the Coprocessor plugin, then both isosurfaces end up being called Contour1 in the output pipeline script. The output script has to be edited by hand to change the name of the second isosurface to Contour2 in order for both isosurfaces to render properly.

6. Finally, the Coprocessor plugin loads correctly with both the client and pvserver processes running on ViewMaster nodes. However, in a distributed mode with the client running on a Viewmaster node and the pvserver running on a Cielo backend node, I could not get both the Local and Remote components of the plugin to successfully load. When I

try to load the Remote component of the Coprocessor plugin on Cielo I get the error message:

ERROR In:

/usr/projects/views/paraview/src/ParaView/4.1.0/ParaView4/ParaViewCore/ClientServerCore/Core/vtkVTPluginLoader.cxx, line 302

vtkVTPluginLoader (0x46cebd0): libvtkpqComponents-pv4.1.so.1: cannot open shared object file: No such file or directory.

I tried to pursue this problem with Alan Scott who is responsible for maintaining the Sandia build of ParaView on Cielo, but unfortunately was not able to get it resolved. This forced me to move data from Cielo to ViewMaster in order to create the pipeline. That should not be necessary if the plugin were working properly.

Problems with the ParaView Tool

I also noted several more general problems with ParaView:

1. Surfaces of constant color that are supposed to be displayed as full intensity white with RGB = (1.0,1.0,1.0) get displayed as off-white instead. For example, in Fig. 5 the isosurface that represents the interface between the plastic and the gas isn't rendered as full intensity white even though that is the color specified for it. It appears as reduced intensity white instead. This seems to be a general problem with ParaView, and I do not think that it is a function of the lighting model. I was not able to find a work-around for the problem which is why the plastic/gas interface in the images of Figs. 5 and 6 appears with a reduced intensity white color.

2. The color map editor seems to have problems with variables having a large dynamic range, a situation I frequently encountered in the current investigation when creating RGB color palettes for my pipelines. Such variables are often not correctly displayed in the color editor, and it is difficult to create new control points in the color map with desired data and color values by clicking in the editor's color bar. It often proved impossible to delete a control point once it was created or to edit its value or color. The way in which I worked around this problem was to create the correct number of control points by clicking in the color bar, then saving out a state file and editing the state file directly to fix the control points in the map by hand. I then restored the modified state file to get the color palette I wanted to create the completed pipeline.

3. I have already noted in Section IV the problem with trying to use volume rendering with the unstructured RAGE mesh and ParaView/Catalyst. ParaView only does volume rendering on a structured grid and there seems to be no way to resample a specified

spatial region of the unstructured RAGE grid to create a distributed structured mesh suitable for scalable volume rendering in ParaView.

```
try: paraview.simple
except: from paraview.simple import *

from paraview import coprocessing
from paraview import simple

from paraview import servermanager

import datetime
current_date = datetime.datetime.now()

write_frequencies    = {'input': [1]}
simulation_input_map = {'pv_output_*': 'input'}

# ---------------------- CoProcessor definition ----------------------

def CreateCoProcessor():
  def _CreatePipeline(coprocessor, datadescription):
    class Pipeline:

      pv_output_ = coprocessor.CreateProducer( datadescription, "input" )
      grid = pv_output_.GetClientSideObject().GetOutputDataObject(0)

      if grid.IsA('vtkImageData') or grid.IsA('vtkUniformGrid'):
        writer = coprocessor.CreateWriter( XMLPImageDataWriter, "grid_%t.pvti", 1 )
      elif grid.IsA('vtkRectilinearGrid'):
        writer = coprocessor.CreateWriter( XMLPRectilinearGridWriter, "grid_%t.pvtr", 1 )
      elif grid.IsA('vtkStructuredGrid'):
        writer = coprocessor.CreateWriter( XMLPStructuredGridWriter, "grid_%t.pvts", 1 )
      elif grid.IsA('vtkPolyData'):
        writer = coprocessor.CreateWriter( XMLPPolyDataWriter, "grid_%t.pvtp", 1 )
      elif grid.IsA('vtkUnstructuredGrid'):
        writer = coprocessor.CreateWriter( XMLPUnstructuredGridWriter, "grid_%t.pvtu", 1 )
      elif grid.IsA('vtkUniformGridAMR'):
        writer = coprocessor.CreateWriter( XMLHierarchicalBoxDataWriter, "grid_%t.vthb", 1 )
      elif grid.IsA('vtkMultiBlockDataSet'):
        writer = coprocessor.CreateWriter( XMLMultiBlockDataWriter, "grid_%t.vtm", 1 )
      else:
        print "Don't know how to create a writer for a ", grid.GetClassName()

    return Pipeline()

  class CoProcessor(coprocessing.CoProcessor):
    def CreatePipeline(self, datadescription):
      self.Pipeline = _CreatePipeline(self, datadescription)

  coprocessor = CoProcessor()
  freqs = {'input': [1]}
  coprocessor.SetUpdateFrequencies(freqs)
  return coprocessor

coprocessor = CreateCoProcessor()
```

Fig. 1(a). Listing of the Python pipeline file “write.py” used with Catalyst in RAGE to generate RAGE mesh data dumps in unstructured VTK format.

```
# ---------------------- Data Selection method ----------------------

def RequestDataDescription(datadescription):
    "Callback to populate the request for current timestep"
    global coprocessor

    if datadescription.GetForceOutput() == True:
        for i in range(datadescription.GetNumberOfInputDescriptions()):
            datadescription.GetInputDescription(i).AllFieldsOn()
            datadescription.GetInputDescription(i).GenerateMeshOn()
        return

    for input_name in simulation_input_map.values():
       coprocessor.LoadRequestedData(datadescription)

# ------------------------ Processing method ----------------------

def DoCoProcessing(datadescription):
    "Callback to do co-processing for current timestep"
    global coprocessor

    timestep = datadescription.GetTimeStep()

    # Update the coprocessor by providing it the newly generated simulation data
    # If the pipeline hasn't been setup yet, this will setup the pipeline.
    coprocessor.UpdateProducers(datadescription)

    # Write output data, if appropriate.
    coprocessor.WriteData(datadescription);

    # Write image capture (Last arg: rescale lookup table)
    #WriteXRAGEImages(datadescription, timestep, False)

    # Live Visualization, if enabled.
    #coprocessor.DoLiveVisualization(datadescription, "localhost", 22222)
```

Fig. 1(b). Listing (continued) of the Python pipeline file “write.py” used with Catalyst in RAGE to generate RAGE mesh data dumps in unstructured VTK format.

```
try: paraview.simple
except: from paraview.simple import *

from paraview import coprocessing
from paraview import simple

from paraview import servermanager

import datetime
current_date = datetime.datetime.now()

write_frequencies    = {'input': [1]}
simulation_input_map = {'pv_output_*': 'input'}

# ---------------------- CoProcessor definition ----------------------

def CreateCoProcessor():
  def _CreatePipeline(coprocessor, datadescription):
    class Pipeline:

      a1_rho_PiecewiseFunction = CreatePiecewiseFunction( Points=[0.0, 0.0, 0.5, 0.0, 3.0, 1.0,
0.5, 0.0] )

      a3_Vorticity_PiecewiseFunction = CreatePiecewiseFunction( Points=[-2e+10, 0.0, 0.5, 0.0,
2e+10, 1.0, 0.5, 0.0] )

      a3_ScalarGradient_PiecewiseFunction = CreatePiecewiseFunction( Points=[5e+13, 0.0, 0.5,
0.0, 5e+17, 1.0, 0.5, 0.0] )

      a1_prs_PiecewiseFunction = CreatePiecewiseFunction( Points=[0.0, 0.0, 0.5, 0.0, 5e+13,
1.0, 0.5, 0.0] )

      a1_rho_PVLookupTable = GetLookupTableForArray( "rho", 1, RGBPoints=[0.0, 0.0, 0.0, 0.0,
0.75, 0.0, 1.0, 1.0, 1.5, 0.0, 1.0, 0.0, 2.25, 1.0, 1.0, 0.0, 3.0, 1.0, 0.0, 0.0],
VectorMode='Magnitude', NanColor=[0.49803900000000001, 0.49803900000000001,
0.49803900000000001], ScalarOpacityFunction=a1_rho_PiecewiseFunction, ColorSpace='RGB',
ScalarRangeInitialized=1.0, LockScalarRange=1 )

      a3_Vorticity_PVLookupTable = GetLookupTableForArray( "Vorticity", 3, RGBPoints=[-2e+10,
0.0, 0.0, 1.0, -1e+10, 0.0, 1.0, 1.0, 0.0, 0.0, 0.0, 0.0, 1e+10, 1.0, 1.0, 0.0, 2e+10, 1.0, 0.0,
0.0], VectorComponent=2, NanColor=[0.49803900000000001, 0.49803900000000001,
0.49803900000000001], ScalarOpacityFunction=a3_Vorticity_PiecewiseFunction, ColorSpace='RGB',
ScalarRangeInitialized=1.0, LockScalarRange=1 )

      a3_ScalarGradient_PVLookupTable = GetLookupTableForArray( "ScalarGradient", 3,
RGBPoints=[5e+13, 0.0, 0.0, 0.0, 5e+14, 0.0, 1.0, 1.0, 5e+15, 0.0, 1.0, 0.0, 5e+16, 1.0, 1.0,
0.0, 5e+17, 1.0, 0.0, 0.0], UseLogScale=1, NanColor=[0.49803900000000001, 0.49803900000000001,
0.49803900000000001], ScalarOpacityFunction=a3_ScalarGradient_PiecewiseFunction,
VectorMode='Magnitude', ColorSpace='RGB', ScalarRangeInitialized=1.0, LockScalarRange=1 )

      a1_prs_PVLookupTable = GetLookupTableForArray( "prs", 1, RGBPoints=[0.0, 0.0, 0.0, 1.0,
1.25e+13, 0.0, 1.0, 1.0, 2.5e+13, 0.0, 1.0, 0.0, 3.75e+13, 1.0, 1.0, 0.0, 5e+13, 1.0, 0.0, 0.0],
VectorMode='Magnitude', NanColor=[0.49803900000000001, 0.49803900000000001,
0.49803900000000001], ScalarOpacityFunction=a1_prs_PiecewiseFunction, ColorSpace='RGB',
ScalarRangeInitialized=1.0, LockScalarRange=1 )
```

Fig. 2(a). Listing of the Python pipeline file "ICFpicmaker.py" used with Catalyst in RAGE to generate the time snapshots of the 2D RAGE simulation shown in Fig. 3.

```
       RenderView2 = coprocessor.CreateView( CreateRenderView, "image_%t.png", 1, 0, 1, 1920,
1024 )
      RenderView2.CameraViewUp = [0.99999837962603078, -0.0018002070194922195, 0.0]
      RenderView2.CacheKey = 0.0
      RenderView2.StereoType = 0
      RenderView2.UseLight = 1
      RenderView2.StereoRender = 0
      RenderView2.CameraPosition = [9.3237904373550998e-05, -0.00098355111510152601,
1.5119130404157799]
      RenderView2.LightSwitch = 0
      RenderView2.OrientationAxesVisibility = 0
      RenderView2.RemoteRenderThreshold = 1.0000000000000001e+299
      RenderView2.CameraClippingRange = [1.496792420011622, 1.5345924810220166]
      RenderView2.LODThreshold = 5.2000000000000002
      RenderView2.InteractionMode = '2D'
      RenderView2.CameraFocalPoint = [9.3237904373550998e-05, -0.00098355111510152601, 0.0]
      RenderView2.CenterAxesVisibility = 0
      RenderView2.CameraParallelScale = 0.0236572935684497
      RenderView2.CenterOfRotation = [0.17499999701976801, 0.0, 0.0]
      RenderView2.StereoCapableWindow = 0

      ScalarBarWidgetRepresentation1 = CreateScalarBar( Title='rho', Position2=[0.13,
0.39224704336399502], Enabled=1, LabelFontSize=12, LookupTable=a1_rho_PVLookupTable,
TitleFontSize=12, Position=[0.055410334346504601, 0.58245729303547999] )
      GetRenderView().Representations.append(ScalarBarWidgetRepresentation1)

      ScalarBarWidgetRepresentation2 = CreateScalarBar( ComponentTitle='Z', Title='Vorticity',
Position2=[0.13, 0.41064388961892301], Enabled=1, LabelFontSize=12,
LookupTable=a3_Vorticity_PVLookupTable, TitleFontSize=12, Position=[0.89806484295846001,
0.58245729303547999] )
      GetRenderView().Representations.append(ScalarBarWidgetRepresentation2)

      ScalarBarWidgetRepresentation3 = CreateScalarBar( Title='Grad P', Position2=[0.13,
0.41064388961892301], Enabled=1, LabelFontSize=12, LookupTable=a3_ScalarGradient_PVLookupTable,
TitleFontSize=12, Position=[0.89806484295846001, 0.045992115637319302] )
      GetRenderView().Representations.append(ScalarBarWidgetRepresentation3)

      ScalarBarWidgetRepresentation4 = CreateScalarBar( Title='prs', Position2=[0.13,
0.41721419185282699], Enabled=1, LabelFontSize=12, LookupTable=a1_prs_PVLookupTable,
TitleFontSize=12, Position=[0.056423505572441697, 0.046320630749014401] )
      GetRenderView().Representations.append(ScalarBarWidgetRepresentation4)

      grid_4515_vtm = coprocessor.CreateProducer( datadescription, "input" )

      CellDatatoPointData2 = CellDatatoPointData( guiName="CellDatatoPointData2", PassCellData=1
)

      Calculator3 = Calculator( guiName="Calculator3", Function='xdt*iHat+ydt*jHat+zdt*kHat',
ResultArrayName='cell_velocity' )

      ComputeDerivatives2 = ComputeDerivatives( guiName="ComputeDerivatives2",
Scalars=['POINTS', 'grd'], Vectors=['POINTS', 'cell_velocity'], OutputTensorType='Nothing',
OutputVectorType='Vorticity' )

      IsoVolume5 = IsoVolume( guiName="IsoVolume5", ThresholdRange=[0.5, 1.0],
InputScalars=['CELLS', 'v02'] )
```

Fig. 2(b). Listing (continued) of the Python pipeline file "ICFpicmaker.py" used with Catalyst in RAGE to generate the time snapshots of the 2D RAGE simulation shown in Fig. 3.

```
      SetActiveSource(CellDatatoPointData2)
      ComputeDerivatives1 = ComputeDerivatives( guiName="ComputeDerivatives1",
Scalars=['POINTS', 'prs'], Vectors=[None, ''], OutputTensorType='Nothing' )

      IsoVolume1 = IsoVolume( guiName="IsoVolume1", ThresholdRange=[0.5, 1.0],
InputScalars=['CELLS', 'v02'] )

      Reflect2 = Reflect( guiName="Reflect2", CopyInput=0 )

      SetActiveSource(grid_4515_vtm)
      Reflect4 = Reflect( guiName="Reflect4", CopyInput=0 )

      SetActiveSource(CellDatatoPointData2)
      Contour1 = Contour( guiName="Contour1", Isosurfaces=[0.5], ContourBy=['POINTS', 'v02'],
PointMergeMethod="Uniform Binning" )

      Reflect1 = Reflect( guiName="Reflect1" )

      SetActiveSource(grid_4515_vtm)
      DataRepresentation1 = Show()
      DataRepresentation1.EdgeColor = [0.0, 0.0, 0.50000762951094835]
      DataRepresentation1.ColorAttributeType = 'CELL_DATA'
      DataRepresentation1.SelectionCellFieldDataArrayName = 'grd'
      DataRepresentation1.ScalarOpacityFunction = a1_rho_PiecewiseFunction
      DataRepresentation1.ColorArrayName = ('CELL_DATA', 'rho')
      DataRepresentation1.ScalarOpacityUnitDistance = 0.0089592792350945
      DataRepresentation1.LookupTable = a1_rho_PVLookupTable
      DataRepresentation1.ExtractedBlockIndex = 1
      DataRepresentation1.ScaleFactor = 0.069999998807907096

      SetActiveSource(CellDatatoPointData2)
      DataRepresentation8 = Show()
      DataRepresentation8.EdgeColor = [0.0, 0.0, 0.50000762951094835]
      DataRepresentation8.ColorAttributeType = 'CELL_DATA'
      DataRepresentation8.SelectionPointFieldDataArrayName = 'grd'
      DataRepresentation8.SelectionCellFieldDataArrayName = 'grd'
      DataRepresentation8.ScalarOpacityFunction = a1_rho_PiecewiseFunction
      DataRepresentation8.ColorArrayName = ('CELL_DATA', 'rho')
      DataRepresentation8.ScalarOpacityUnitDistance = 0.0089592792350945
      DataRepresentation8.Visibility = 0
      DataRepresentation8.LookupTable = a1_rho_PVLookupTable
      DataRepresentation8.ExtractedBlockIndex = 1
      DataRepresentation8.ScaleFactor = 0.069999998807907096

      SetActiveSource(Calculator3)
      DataRepresentation9 = Show()
      DataRepresentation9.EdgeColor = [0.0, 0.0, 0.50000762951094835]
      DataRepresentation9.ColorAttributeType = 'CELL_DATA'
      DataRepresentation9.SelectionPointFieldDataArrayName = 'cell_velocity'
      DataRepresentation9.SelectionCellFieldDataArrayName = 'grd'
      DataRepresentation9.ScalarOpacityFunction = a1_rho_PiecewiseFunction
      DataRepresentation9.ColorArrayName = ('CELL_DATA', 'rho')
      DataRepresentation9.ScalarOpacityUnitDistance = 0.0089592792350945
      DataRepresentation9.Visibility = 0
      DataRepresentation9.LookupTable = a1_rho_PVLookupTable
      DataRepresentation9.ExtractedBlockIndex = 1
      DataRepresentation9.ScaleFactor = 0.069999998807907096
```

Fig. 2(c). Listing (continued) of the Python pipeline file "ICFpicmaker.py" used with Catalyst in RAGE to generate the time snapshots of the 2D RAGE simulation shown in Fig. 3.

```
SetActiveSource(ComputeDerivatives2)
DataRepresentation10 = Show()
DataRepresentation10.EdgeColor = [0.0, 0.0, 0.50000762951094835]
DataRepresentation10.ColorAttributeType = 'CELL_DATA'
DataRepresentation10.SelectionPointFieldDataArrayName = 'cell_velocity'
DataRepresentation10.SelectionCellFieldDataArrayName = 'grd'
DataRepresentation10.ScalarOpacityFunction = a3_Vorticity_PiecewiseFunction
DataRepresentation10.ColorArrayName = ('CELL_DATA', 'Vorticity')
DataRepresentation10.ScalarOpacityUnitDistance = 0.0089592792350945
DataRepresentation10.Visibility = 0
DataRepresentation10.LookupTable = a3_Vorticity_PVLookupTable
DataRepresentation10.ExtractedBlockIndex = 1
DataRepresentation10.ScaleFactor = 0.069999998807907096

SetActiveSource(IsoVolume5)
DataRepresentation11 = Show()
DataRepresentation11.EdgeColor = [0.0, 0.0, 0.50000762951094835]
DataRepresentation11.ColorAttributeType = 'CELL_DATA'
DataRepresentation11.SelectionPointFieldDataArrayName = 'cell_velocity'
DataRepresentation11.SelectionCellFieldDataArrayName = 'grd'
DataRepresentation11.ScalarOpacityFunction = a3_Vorticity_PiecewiseFunction
DataRepresentation11.ColorArrayName = ('CELL_DATA', 'Vorticity')
DataRepresentation11.ScalarOpacityUnitDistance = 0.00059658224132210402
DataRepresentation11.LookupTable = a3_Vorticity_PVLookupTable
DataRepresentation11.ExtractedBlockIndex = 1
DataRepresentation11.ScaleFactor = 0.0039843749254941897

SetActiveSource(ComputeDerivatives1)
DataRepresentation1 = Show()
DataRepresentation1.EdgeColor = [0.0, 0.0, 0.50000762951094835]
DataRepresentation1.ColorAttributeType = 'CELL_DATA'
DataRepresentation1.SelectionPointFieldDataArrayName = 'grd'
DataRepresentation1.SelectionCellFieldDataArrayName = 'grd'
DataRepresentation1.ScalarOpacityFunction = a1_rho_PiecewiseFunction
DataRepresentation1.ColorArrayName = ('CELL_DATA', 'rho')
DataRepresentation1.ScalarOpacityUnitDistance = 0.0089592792350945
DataRepresentation1.Visibility = 0
DataRepresentation1.LookupTable = a1_rho_PVLookupTable
DataRepresentation1.ExtractedBlockIndex = 1
DataRepresentation1.ScaleFactor = 0.069999998807907096

SetActiveSource(IsoVolume1)
DataRepresentation2 = Show()
DataRepresentation2.EdgeColor = [0.0, 0.0, 0.50000762951094835]
DataRepresentation2.ColorAttributeType = 'CELL_DATA'
DataRepresentation2.SelectionPointFieldDataArrayName = 'grd'
DataRepresentation2.SelectionCellFieldDataArrayName = 'grd'
DataRepresentation2.ScalarOpacityFunction = a1_rho_PiecewiseFunction
DataRepresentation2.ColorArrayName = ('CELL_DATA', 'rho')
DataRepresentation2.ScalarOpacityUnitDistance = 0.00059658224132210402
DataRepresentation2.Visibility = 0
DataRepresentation2.LookupTable = a1_rho_PVLookupTable
DataRepresentation2.ExtractedBlockIndex = 1
DataRepresentation2.ScaleFactor = 0.0039843749254941897
```

Fig. 2(d). Listing (continued) of the Python pipeline file "ICFpicmaker.py" used with Catalyst in RAGE to generate the time snapshots of the 2D RAGE simulation shown in Fig. 3.

```
SetActiveSource(Reflect2)
DataRepresentation4 = Show()
DataRepresentation4.EdgeColor = [0.0, 0.0, 0.50000762951094835]
DataRepresentation4.ColorAttributeType = 'CELL_DATA'
DataRepresentation4.SelectionPointFieldDataArrayName = 'grd'
DataRepresentation4.SelectionCellFieldDataArrayName = 'grd'
DataRepresentation4.ScalarOpacityFunction = a3_ScalarGradient_PiecewiseFunction
DataRepresentation4.ColorArrayName = ('CELL_DATA', 'ScalarGradient')
DataRepresentation4.ScalarOpacityUnitDistance = 0.00059658224132210402
DataRepresentation4.LookupTable = a3_ScalarGradient_PVLookupTable
DataRepresentation4.ExtractedBlockIndex = 1
DataRepresentation4.Position = [0.0, 0.0, 9.9999999999999995e-07]
DataRepresentation4.ScaleFactor = 0.0039843749254941897

SetActiveSource(Reflect4)
DataRepresentation6 = Show()
DataRepresentation6.EdgeColor = [0.0, 0.0, 0.50000762951094835]
DataRepresentation6.ColorAttributeType = 'CELL_DATA'
DataRepresentation6.SelectionCellFieldDataArrayName = 'grd'
DataRepresentation6.ScalarOpacityFunction = a1_prs_PiecewiseFunction
DataRepresentation6.ColorArrayName = ('CELL_DATA', 'prs')
DataRepresentation6.ScalarOpacityUnitDistance = 0.0089592792350945
DataRepresentation6.LookupTable = a1_prs_PVLookupTable
DataRepresentation6.ExtractedBlockIndex = 1
DataRepresentation6.ScaleFactor = 0.069999998807907096

SetActiveSource(Contour1)
DataRepresentation1 = Show()
DataRepresentation1.EdgeColor = [0.0, 0.0, 0.50000762951094835]
DataRepresentation1.ColorAttributeType = 'CELL_DATA'
DataRepresentation1.SelectionPointFieldDataArrayName = 'grd'
DataRepresentation1.SelectionCellFieldDataArrayName = 'grd'
DataRepresentation1.Visibility = 0
DataRepresentation1.LookupTable = a1_rho_PVLookupTable
DataRepresentation1.ScaleFactor = 0.0039821326732635502

SetActiveSource(Reflect1)
DataRepresentation2 = Show()
DataRepresentation2.EdgeColor = [0.0, 0.0, 0.50000762951094835]
DataRepresentation2.SelectionPointFieldDataArrayName = 'grd'
DataRepresentation2.SelectionCellFieldDataArrayName = 'grd'
DataRepresentation2.ScalarOpacityUnitDistance = 0.0033510525697585656
DataRepresentation2.ExtractedBlockIndex = 1
DataRepresentation2.ScaleFactor = 0.0039821326732635502

annotationColor = [1.0, 1.0, 1.0]
VersionText = Text(Text='2D Rage Omega Capsule P30 Asymmetry 50%')
VersionRep = Show()
VersionRep.WindowLocation = 'UpperCenter'
VersionRep.Color = annotationColor
VersionRep.FontSize = 10
VersionRep.Orientation = 0
VersionRep.TextScaleMode = 'Viewport'
```

Fig. 2(e). Listing (continued) of the Python pipeline file "ICFpicmaker.py" used with Catalyst in RAGE to generate the time snapshots of the 2D RAGE simulation shown in Fig. 3.

```
      AnnotateTimeFilter1 = AnnotateTimeFilter()
      TimeRep = Show()
      AnnotateTimeFilter1.Format = 'time = %10.4e s'
      TimeRep.WindowLocation = 'LowerLeftCorner'
      TimeRep.Color = annotationColor
      TimeRep.FontSize = 10
      TimeRep.TextScaleMode = 'Viewport'

    return Pipeline()

  class CoProcessor(coprocessing.CoProcessor):
    def CreatePipeline(self, datadescription):
      self.Pipeline = _CreatePipeline(self, datadescription)

  coprocessor = CoProcessor()
  freqs = {'input': [1]}
  coprocessor.SetUpdateFrequencies(freqs)
  return coprocessor

coprocessor = CreateCoProcessor()

# ---------------------- Data Selection method ----------------------

def RequestDataDescription(datadescription):
    "Callback to populate the request for current timestep"
    global coprocessor

    if datadescription.GetForceOutput() == True:
        for i in range(datadescription.GetNumberOfInputDescriptions()):
            datadescription.GetInputDescription(i).AllFieldsOn()
            datadescription.GetInputDescription(i).GenerateMeshOn()
        return

    for input_name in simulation_input_map.values():
       coprocessor.LoadRequestedData(datadescription)

# ------------------------ Processing method ------------------------

def DoCoProcessing(datadescription):
    "Callback to do co-processing for current timestep"
    global coprocessor

    timestep = datadescription.GetTimeStep()

    # Update the coprocessor by providing it the newly generated simulation data
    # If the pipeline hasn't been setup yet, this will setup the pipeline.
    coprocessor.UpdateProducers(datadescription)

    # Write output data, if appropriate.
    #coprocessor.WriteData(datadescription);

    # Write image capture (Last arg: rescale lookup table), if appropriate.
    #coprocessor.WriteImages(datadescription, rescale_lookuptable=False)
    WriteXRAGEImages(datadescription, timestep)

    # Live Visualization, if enabled.
    #coprocessor.DoLiveVisualization(datadescription, "localhost", 22222)
```

Fig. 2(f). Listing (continued) of the Python pipeline file "ICFpicmaker.py" used with Catalyst in RAGE to generate the time snapshots of the 2D RAGE simulation shown in Fig. 3.

```
# ------------------------ Write images ----------------------------------------

def WriteXRAGEImages(datadescription, timestep):

  grid = datadescription.GetInputDescriptionByName("input").GetGrid()

  for view in servermanager.GetRenderViews():
    if (timestep % view.cpFrequency == 0 or
        datadescription.GetForceOutput() == True):

        # Automatic camera bounds
        cameraName = 'v02' + "_camera"
        cameraArray = grid.GetFieldData().GetArray(cameraName)
        cameraBounds = cameraArray.GetTuple(0)
        view.SMProxy.ResetCamera(cameraBounds)
        xmax = 0.45*max(abs(cameraBounds[0]),abs(cameraBounds[1]))
        cameraBounds2 = (-xmax,xmax,cameraBounds[2],cameraBounds[3],cameraBounds[4],
cameraBounds[5])
        view.SMProxy.ResetCamera(cameraBounds2)

        # Output file name
        fileName = "pv_%05i.png"
        fileName = fileName % timestep

        view.ViewTime = datadescription.GetTime()
        Render(view)
        WriteImage(fileName, view, Magnification=view.cpMagnification)
```

Fig. 2(g). Listing (continued) of the Python pipeline file "ICFpicmaker.py" used with Catalyst in RAGE to generate the time snapshots of the 2D RAGE simulation shown in Fig. 3.

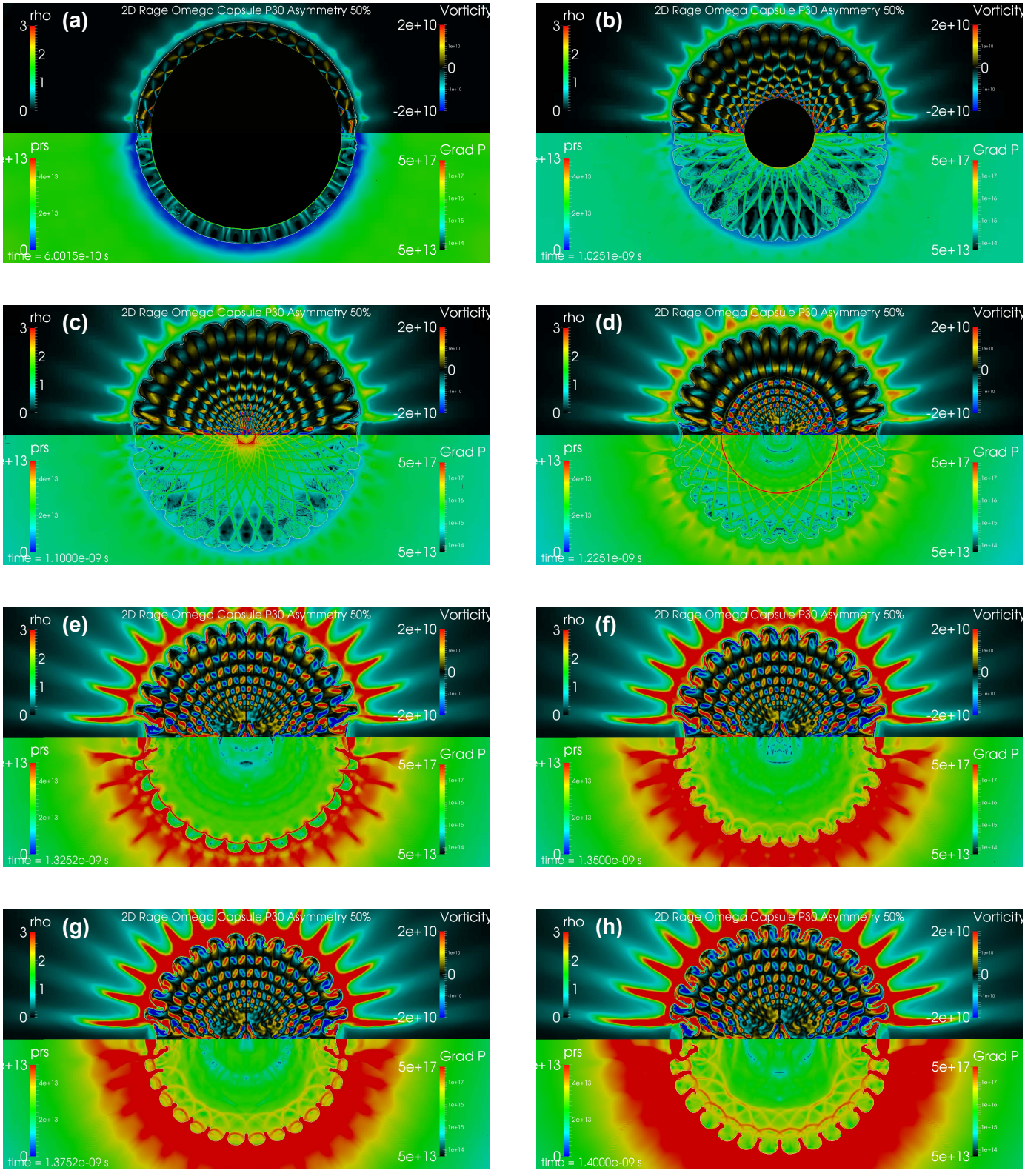

Fig. 3. Eight time snapshots from 2D RAGE simulation of the P30 Omega capsule generated using the in-situ visualization capabilities of RAGE. In the lower portion of each panel the plastic shell is colored by pressure while the gas is colored by grad P. In the upper portion the plastic shell is colored by density while the gas is colored by azimuthal vorticity.

```
try: paraview.simple
except: from paraview.simple import *

from paraview import coprocessing
from paraview import simple

from paraview import servermanager

import datetime
current_date = datetime.datetime.now()

write_frequencies    = {'input': [1]}
simulation_input_map = {'pv_output_*': 'input'}

# ---------------------- CoProcessor definition ----------------------

def CreateCoProcessor():
  def _CreatePipeline(coprocessor, datadescription):
    class Pipeline:
      a1_zdt_PiecewiseFunction = CreatePiecewiseFunction( Points=[-27098332.0, 0.0, 0.5, 0.0,
34138856.0, 1.0, 0.5, 0.0] )

      a3_Vorticity_PiecewiseFunction = CreatePiecewiseFunction( Points=[-2.5e11, 0.0, 0.5, 0.0,
2.5e11, 1.0, 0.5, 0.0] )

      a1_v02_PiecewiseFunction = CreatePiecewiseFunction( Points=[0.0, 0.0, 0.5, 0.0, 1.0, 1.0,
0.5, 0.0] )

      a3_ScalarGradient_PiecewiseFunction = CreatePiecewiseFunction( Points=[2e14, 0.0, 0.5,
0.0, 2e+18, 1.0, 0.5, 0.0] )

      a1_MagVort_PiecewiseFunction = CreatePiecewiseFunction( Points=[0.0, 0.0, 0.5, 0.0,
131250242310.37444, 1.0, 0.5, 0.0] )

      a3_GasVort_PiecewiseFunction = CreatePiecewiseFunction( Points=[-5e11, 0.0, 0.5, 0.0,
5e11, 1.0, 0.5, 0.0] )

      a1_prs_PiecewiseFunction = CreatePiecewiseFunction( Points=[8467661.0, 0.0, 0.5, 0.0,
752997290213376.0, 1.0, 0.5, 0.0] )

      a1_rho_PiecewiseFunction = CreatePiecewiseFunction( Points=[0.0023837601765990301, 0.0,
0.5, 0.0, 11.718677520751999, 1.0, 0.5, 0.0] )

      a1_xdt_PiecewiseFunction = CreatePiecewiseFunction( Points=[-27098332.0, 0.0, 0.5, 0.0,
34138780.0, 1.0, 0.5, 0.0] )

      a1_ydt_PiecewiseFunction = CreatePiecewiseFunction( Points=[-34075808.0, 0.0, 0.5, 0.0,
36782544.0, 1.0, 0.5, 0.0] )

      a1_zdt_PVLookupTable = GetLookupTableForArray( "zdt", 1, RGBPoints=[-27098332.0, 0.0, 0.0,
1.0, 34138856.0, 1.0, 0.0, 0.0], VectorMode='Magnitude', NanColor=[0.49803900000000001,
0.49803900000000001, 0.49803900000000001], ScalarOpacityFunction=a1_zdt_PiecewiseFunction,
ColorSpace='HSV', ScalarRangeInitialized=1.0 )
```

Fig. 4(a). Listing of the Python pipeline file "VortTubes.py" used with Catalyst in RAGE to generate the time snapshots of the 3D RAGE simulation shown in Fig. 5.

```
      a3_Vorticity_PVLookupTable = GetLookupTableForArray( "Vorticity", 3, RGBPoints=[-2.5e11,
0.0, 0.0, 1.0, -1.25e11, 0.0, 1.0, 1.0, 0.0, 0.0, 0.0, 0.0, 1.25e11, 1.0, 1.0, 0.0, 2.5e11, 1.0,
0.0, 0.0], VectorComponent=2, NanColor=[0.49803900000000001, 0.49803900000000001,
0.49803900000000001], ScalarOpacityFunction=a3_Vorticity_PiecewiseFunction, ColorSpace='RGB',
ScalarRangeInitialized=1.0, LockScalarRange=1 )

      a1_v02_PVLookupTable = GetLookupTableForArray( "v02", 1, RGBPoints=[0.0, 0.0, 0.0, 1.0,
1.0, 1.0, 0.0, 0.0], VectorMode='Magnitude', NanColor=[0.49803900000000001, 0.49803900000000001,
0.49803900000000001], ScalarOpacityFunction=a1_v02_PiecewiseFunction, ColorSpace='HSV',
ScalarRangeInitialized=1.0 )

      a3_ScalarGradient_PVLookupTable = GetLookupTableForArray( "ScalarGradient", 3,
RGBPoints=[200000000000000.0, 0.0, 0.0, 1.0, 2000000000000000.0, 0.0, 1.0, 1.0,
20000000000000000.0, 0.0, 1.0, 0.0, 2e+17, 1.0, 1.0, 0.0, 2e+18, 1.0, 0.0, 0.0], UseLogScale=1,
NanColor=[0.49803900000000001, 0.49803900000000001, 0.49803900000000001],
ScalarOpacityFunction=a3_ScalarGradient_PiecewiseFunction, ColorSpace='RGB',
VectorMode='Magnitude', ScalarRangeInitialized=1.0, LockScalarRange=1 )

      a1_MagVort_PVLookupTable = GetLookupTableForArray( "MagVort", 1, RGBPoints=[0.0, 0.0, 0.0,
1.0, 131250242310.37444, 1.0, 0.0, 0.0], VectorMode='Magnitude', NanColor=[0.49803900000000001,
0.49803900000000001, 0.49803900000000001], ScalarOpacityFunction=a1_MagVort_PiecewiseFunction,
ColorSpace='RGB', ScalarRangeInitialized=1.0 )

      a3_GasVort_PVLookupTable = GetLookupTableForArray( "GasVort", 3, RGBPoints=[-5e11, 0.0,
0.0, 1.0, -2.5e11, 0.0, 1.0, 1.0, 0.0, 0.80000000000000004, 0.80000000000000004,
0.80000000000000004, 2.5e11, 1.0, 1.0, 0.0, 5e11, 1.0, 0.0, 0.0], VectorComponent=1,
NanColor=[0.49803900000000001, 0.49803900000000001, 0.49803900000000001],
ScalarOpacityFunction=a3_GasVort_PiecewiseFunction, ColorSpace='RGB',
ScalarRangeInitialized=1.0, LockScalarRange=1 )

      a1_prs_PVLookupTable = GetLookupTableForArray( "prs", 1, RGBPoints=[8467661.0, 0.0, 0.0,
1.0, 752997290213376.0, 1.0, 0.0, 0.0], VectorMode='Magnitude', NanColor=[0.49803900000000001,
0.49803900000000001, 0.49803900000000001], ScalarOpacityFunction=a1_prs_PiecewiseFunction,
ColorSpace='HSV', ScalarRangeInitialized=1.0 )

      a1_rho_PVLookupTable = GetLookupTableForArray( "rho", 1, RGBPoints=[0.0023837601765990301,
0.0, 0.0, 1.0, 11.718677520751999, 1.0, 0.0, 0.0], VectorMode='Magnitude',
NanColor=[0.49803900000000001, 0.49803900000000001, 0.49803900000000001],
ScalarOpacityFunction=a1_rho_PiecewiseFunction, ColorSpace='HSV', ScalarRangeInitialized=1.0 )

      a1_xdt_PVLookupTable = GetLookupTableForArray( "xdt", 1, RGBPoints=[-27098332.0, 0.0, 0.0,
1.0, 34138780.0, 1.0, 0.0, 0.0], VectorMode='Magnitude', NanColor=[0.49803900000000001,
0.49803900000000001, 0.49803900000000001], ScalarOpacityFunction=a1_xdt_PiecewiseFunction,
ColorSpace='HSV', ScalarRangeInitialized=1.0 )

      a1_ydt_PVLookupTable = GetLookupTableForArray( "ydt", 1, RGBPoints=[-34075808.0, 0.0, 0.0,
1.0, 36782544.0, 1.0, 0.0, 0.0], VectorMode='Magnitude', NanColor=[0.49803900000000001,
0.49803900000000001, 0.49803900000000001], ScalarOpacityFunction=a1_ydt_PiecewiseFunction,
ColorSpace='HSV', ScalarRangeInitialized=1.0 )
```

Fig. 4(b). Listing (continued) of the Python pipeline file "VortTubes.py" used with Catalyst in RAGE to generate the time snapshots of the 3D RAGE simulation shown in Fig. 5.

```
      RenderView2 = coprocessor.CreateView( CreateRenderView, "image_%t.png", 1, 0, 1, 1920,
1024 )
      RenderView2.CameraViewUp = [0.91493507046793898, -0.32880673441105501,
0.23405116584577301]
      RenderView2.CacheKey = 0.0
      RenderView2.StereoType = 0
      RenderView2.UseLight = 1
      RenderView2.StereoRender = 0
      RenderView2.CameraPosition = [0.015960775082022899, 0.026938869311035001, -
0.016873513743119101]
      RenderView2.LightSwitch = 0
      RenderView2.OrientationAxesVisibility = 0
      RenderView2.RemoteRenderThreshold = 1.0000000000000001e+299
      RenderView2.CameraClippingRange = [0.0071085153587034725, 0.057971252754051933]
      RenderView2.LODThreshold = 5.2000000000000002
      RenderView2.CameraFocalPoint = [-0.17357535532339599, -0.33301768236438201,
0.21836186457037501]
      RenderView2.CenterAxesVisibility = 0
      RenderView2.CameraParallelScale = 0.121625220712545
      RenderView2.CenterOfRotation = [7.4505805969238298e-09, 0.0049855625256896002,
0.0052203135564923304]
      RenderView2.StereoCapableWindow = 0

      ScalarBarWidgetRepresentation1 = CreateScalarBar( ComponentTitle='Magnitude', Title='Grad
P', Position2=[0.13, 0.34625492772667499], Enabled=1, LabelFontSize=12,
LookupTable=a3_ScalarGradient_PVLookupTable, TitleFontSize=12, Position=[0.897,
0.014783180026281199] )
      GetRenderView().Representations.append(ScalarBarWidgetRepresentation1)

      ScalarBarWidgetRepresentation2 = CreateScalarBar( ComponentTitle='Z', Title='GasVort',
Position2=[0.13, 0.33574244415243099], Enabled=1, LabelFontSize=12,
LookupTable=a3_Vorticity_PVLookupTable, TitleFontSize=12, Position=[0.86, 0.632345597897503] )
      GetRenderView().Representations.append(ScalarBarWidgetRepresentation2)

      ScalarBarWidgetRepresentation1 = CreateScalarBar( ComponentTitle='Y', Title='GasVort',
Position2=[0.13, 0.33574244415243099], Enabled=1, LabelFontSize=12,
LookupTable=a3_GasVort_PVLookupTable, TitleFontSize=12, Position=[0.10832384067782699,
0.632345597897503] )
      GetRenderView().Representations.append(ScalarBarWidgetRepresentation1)

      grid_0_vtm = coprocessor.CreateProducer( datadescription, "input" )

      CellDatatoPointData1 = CellDatatoPointData( guiName="CellDatatoPointData1" )

      Calculator1 = Calculator( guiName="Calculator1", Function='iHat*xdt+jHat*ydt+kHat*zdt',
ResultArrayName='cell_velocity_pn' )

      ComputeDerivatives2 = ComputeDerivatives( guiName="ComputeDerivatives2",
Scalars=['POINTS', 'prs'], Vectors=['POINTS', 'cell_velocity_pn'], OutputTensorType='Nothing',
OutputVectorType='Vorticity' )

      Slice6 = Slice( guiName="Slice6", SliceOffsetValues=[0.0], SliceType="Plane" )
      Slice6.SliceType.Origin = [0.050000000745058101, 0.050000000745058101,
7.4505810000000002e-10]
      Slice6.SliceType.Normal = [0.0, 0.0, 1.0]

      IsoVolume4 = IsoVolume( guiName="IsoVolume4", ThresholdRange=[0.5, 1.01],
InputScalars=['POINTS', 'v02'] )
```

Fig. 4(c). Listing (continued) of the Python pipeline file "VortTubes.py" used with Catalyst in RAGE to generate the time snapshots of the 3D RAGE simulation shown in Fig. 5.

```
       SetActiveSource(CellDatatoPointData1)
      Contour1 = Contour( guiName="Contour1", Isosurfaces=[0.5], ContourBy=['POINTS', 'v02'],
PointMergeMethod="Uniform Binning" )

      SetActiveSource(CellDatatoPointData1)
      ComputeDerivatives3 = ComputeDerivatives( guiName="ComputeDerivatives3",
Scalars=['POINTS', 'prs'], Vectors=[None, ''], OutputTensorType='Nothing' )

      Slice7 = Slice( guiName="Slice7", SliceOffsetValues=[0.0], SliceType="Plane" )
      Slice7.SliceType.Origin = [7.4505810000000005e-09, 0.050000000745058101,
0.050000000745058101]

      IsoVolume5 = IsoVolume( guiName="IsoVolume5", ThresholdRange=[0.5, 1.01],
InputScalars=['POINTS', 'v02'] )

      Reflect1 = Reflect( guiName="Reflect1", CopyInput=0, Plane='Z Min' )

      SetActiveSource(ComputeDerivatives2)
      CellDatatoPointData1 = CellDatatoPointData( guiName="CellDatatoPointData1" )

      SetActiveSource(CellDatatoPointData1)
      Calculator1 = Calculator( guiName="Calculator1", Function='Vorticity',
ResultArrayName='GasVort' )

      SetActiveSource(Calculator1)
      Calculator2 = Calculator( guiName="Calculator2", Function='mag(GasVort)',
ResultArrayName='MagVort' )

      Contour2 = Contour( guiName="Contour2", Isosurfaces=[2.5e11], ContourBy=['POINTS',
'MagVort'], PointMergeMethod="Uniform Binning" )

      Reflect2 = Reflect( guiName="Reflect2", CopyInput=0, Plane='Z Min' )

      SetActiveSource(grid_0_vtm)
      DataRepresentation1 = Show()
      DataRepresentation1.EdgeColor = [0.0, 0.0, 0.50000762951094835]
      DataRepresentation1.ColorAttributeType = 'CELL_DATA'
      DataRepresentation1.SelectionCellFieldDataArrayName = 'grd'
      DataRepresentation1.ScalarOpacityFunction = a1_zdt_PiecewiseFunction
      DataRepresentation1.ColorArrayName = ('CELL_DATA', 'zdt')
      DataRepresentation1.ScalarOpacityUnitDistance = 0.00036236486270562699
      DataRepresentation1.Visibility = 0
      DataRepresentation1.LookupTable = a1_zdt_PVLookupTable
      DataRepresentation1.ExtractedBlockIndex = 1
      DataRepresentation1.ScaleFactor = 0.0100000001490116

      SetActiveSource(CellDatatoPointData1)
      DataRepresentation5 = Show()
      DataRepresentation5.EdgeColor = [0.0, 0.0, 0.50000762951094835]
      DataRepresentation5.SelectionPointFieldDataArrayName = 'grd'
      DataRepresentation5.SelectionCellFieldDataArrayName = 'grd'
      DataRepresentation5.ScalarOpacityUnitDistance = 0.00036236486270562699
      DataRepresentation5.Visibility = 0
      DataRepresentation5.ExtractedBlockIndex = 1
      DataRepresentation5.ScaleFactor = 0.0100000001490116
```

Fig. 4(d). Listing (continued) of the Python pipeline file "VortTubes.py" used with Catalyst in RAGE to generate the time snapshots of the 3D RAGE simulation shown in Fig. 5.

```
SetActiveSource(Calculator1)
DataRepresentation6 = Show()
DataRepresentation6.EdgeColor = [0.0, 0.0, 0.50000762951094835]
DataRepresentation6.SelectionPointFieldDataArrayName = 'cell_velocity_pn'
DataRepresentation6.SelectionCellFieldDataArrayName = 'grd'
DataRepresentation6.ScalarOpacityUnitDistance = 0.00036236486270562699
DataRepresentation6.Visibility = 0
DataRepresentation6.ExtractedBlockIndex = 1
DataRepresentation6.ScaleFactor = 0.0100000001490116

SetActiveSource(ComputeDerivatives2)
DataRepresentation7 = Show()
DataRepresentation7.EdgeColor = [0.0, 0.0, 0.50000762951094835]
DataRepresentation7.ColorAttributeType = 'CELL_DATA'
DataRepresentation7.SelectionPointFieldDataArrayName = 'cell_velocity_pn'
DataRepresentation7.SelectionCellFieldDataArrayName = 'Vorticity'
DataRepresentation7.ScalarOpacityFunction = a3_Vorticity_PiecewiseFunction
DataRepresentation7.ColorArrayName = ('CELL_DATA', 'Vorticity')
DataRepresentation7.ScalarOpacityUnitDistance = 0.00036236486270562699
DataRepresentation7.Visibility = 0
DataRepresentation7.LookupTable = a3_Vorticity_PVLookupTable
DataRepresentation7.ExtractedBlockIndex = 1
DataRepresentation7.ScaleFactor = 0.0100000001490116

SetActiveSource(Slice6)
DataRepresentation11 = Show()
DataRepresentation11.EdgeColor = [0.0, 0.0, 0.50000762951094835]
DataRepresentation11.SelectionPointFieldDataArrayName = 'cell_velocity_pn'
DataRepresentation11.SelectionCellFieldDataArrayName = 'Vorticity'
DataRepresentation11.ColorArrayName = ('POINT_DATA', 'v02')
DataRepresentation11.Visibility = 0
DataRepresentation11.LookupTable = a1_v02_PVLookupTable
DataRepresentation11.ScaleFactor = 0.0100000001490116

SetActiveSource(IsoVolume4)
DataRepresentation14 = Show()
DataRepresentation14.EdgeColor = [0.0, 0.0, 0.50000762951094835]
DataRepresentation14.ColorAttributeType = 'CELL_DATA'
DataRepresentation14.SelectionPointFieldDataArrayName = 'cell_velocity_pn'
DataRepresentation14.SelectionCellFieldDataArrayName = 'Vorticity'
DataRepresentation14.ScalarOpacityFunction = a3_Vorticity_PiecewiseFunction
DataRepresentation14.ColorArrayName = ('CELL_DATA', 'Vorticity')
DataRepresentation14.ScalarOpacityUnitDistance = 0.00019206879640315999
DataRepresentation14.LookupTable = a3_Vorticity_PVLookupTable
DataRepresentation14.ExtractedBlockIndex = 1
DataRepresentation14.ScaleFactor = 0.0010440627112984699

SetActiveSource(Contour1)
DataRepresentation15 = Show()
DataRepresentation15.EdgeColor = [0.0, 0.0, 0.50000762951094835]
DataRepresentation15.SelectionPointFieldDataArrayName = 'grd'
DataRepresentation15.DiffuseColor = [1.0, 1.0, 1.0]
DataRepresentation15.SelectionCellFieldDataArrayName = 'grd'
DataRepresentation15.ScaleFactor = 0.0010440627112984699
```

Fig. 4(e). Listing (continued) of the Python pipeline file "VortTubes.py" used with Catalyst in RAGE to generate the time snapshots of the 3D RAGE simulation shown in Fig. 5.

```
SetActiveSource(ComputeDerivatives3)
DataRepresentation16 = Show()
DataRepresentation16.EdgeColor = [0.0, 0.0, 0.50000762951094835]
DataRepresentation16.ColorAttributeType = 'CELL_DATA'
DataRepresentation16.SelectionPointFieldDataArrayName = 'grd'
DataRepresentation16.SelectionCellFieldDataArrayName = 'ScalarGradient'
DataRepresentation16.ScalarOpacityFunction = a3_ScalarGradient_PiecewiseFunction
DataRepresentation16.ColorArrayName = ('CELL_DATA', 'ScalarGradient')
DataRepresentation16.ScalarOpacityUnitDistance = 0.00036236486270562699
DataRepresentation16.Visibility = 0
DataRepresentation16.LookupTable = a3_ScalarGradient_PVLookupTable
DataRepresentation16.ExtractedBlockIndex = 1
DataRepresentation16.ScaleFactor = 0.0100000001490116

SetActiveSource(Slice7)
DataRepresentation17 = Show()
DataRepresentation17.EdgeColor = [0.0, 0.0, 0.50000762951094835]
DataRepresentation17.ColorAttributeType = 'CELL_DATA'
DataRepresentation17.SelectionPointFieldDataArrayName = 'grd'
DataRepresentation17.SelectionCellFieldDataArrayName = 'ScalarGradient'
DataRepresentation17.ColorArrayName = ('CELL_DATA', 'ScalarGradient')
DataRepresentation17.Visibility = 0
DataRepresentation17.LookupTable = a3_ScalarGradient_PVLookupTable
DataRepresentation17.ScaleFactor = 0.0100000001490116

SetActiveSource(IsoVolume5)
DataRepresentation18 = Show()
DataRepresentation18.EdgeColor = [0.0, 0.0, 0.50000762951094835]
DataRepresentation18.ColorAttributeType = 'CELL_DATA'
DataRepresentation18.SelectionPointFieldDataArrayName = 'grd'
DataRepresentation18.SelectionCellFieldDataArrayName = 'ScalarGradient'
DataRepresentation18.ScalarOpacityFunction = a3_ScalarGradient_PiecewiseFunction
DataRepresentation18.ColorArrayName = ('CELL_DATA', 'ScalarGradient')
DataRepresentation18.ScalarOpacityUnitDistance = 0.00019109952233533701
DataRepresentation18.Visibility = 0
DataRepresentation18.LookupTable = a3_ScalarGradient_PVLookupTable
DataRepresentation18.ExtractedBlockIndex = 1
DataRepresentation18.ScaleFactor = 0.0010440627112984699

SetActiveSource(Reflect1)
DataRepresentation19 = Show()
DataRepresentation19.EdgeColor = [0.0, 0.0, 0.50000762951094835]
DataRepresentation19.ColorAttributeType = 'CELL_DATA'
DataRepresentation19.SelectionPointFieldDataArrayName = 'grd'
DataRepresentation19.SelectionCellFieldDataArrayName = 'ScalarGradient'
DataRepresentation19.ScalarOpacityFunction = a3_ScalarGradient_PiecewiseFunction
DataRepresentation19.ColorArrayName = ('CELL_DATA', 'ScalarGradient')
DataRepresentation19.ScalarOpacityUnitDistance = 0.00019109952233533701
DataRepresentation19.LookupTable = a3_ScalarGradient_PVLookupTable
DataRepresentation19.ExtractedBlockIndex = 1
DataRepresentation19.ScaleFactor = 0.0010440627112984699
```

Fig. 4(f). Listing (continued) of the Python pipeline file "VortTubes.py" used with Catalyst in RAGE to generate the time snapshots of the 3D RAGE simulation shown in Fig. 5.

```
SetActiveSource(CellDatatoPointData1)
DataRepresentation1 = Show()
DataRepresentation1.EdgeColor = [0.0, 0.0, 0.50000762951094835]
DataRepresentation1.SelectionPointFieldDataArrayName = 'cell_velocity_pn'
DataRepresentation1.SelectionCellFieldDataArrayName = 'Vorticity'
DataRepresentation1.ScalarOpacityUnitDistance = 0.00036236486270562699
DataRepresentation1.Visibility = 0
DataRepresentation1.ExtractedBlockIndex = 1
DataRepresentation1.ScaleFactor = 0.0100000001490116

SetActiveSource(Calculator1)
DataRepresentation2 = Show()
DataRepresentation2.EdgeColor = [0.0, 0.0, 0.50000762951094835]
DataRepresentation2.SelectionPointFieldDataArrayName = 'cell_velocity_pn'
DataRepresentation2.SelectionCellFieldDataArrayName = 'Vorticity'
DataRepresentation2.ScalarOpacityUnitDistance = 0.00036236486270562699
DataRepresentation2.Visibility = 0
DataRepresentation2.ExtractedBlockIndex = 1
DataRepresentation2.ScaleFactor = 0.0100000001490116

SetActiveSource(Calculator2)
DataRepresentation3 = Show()
DataRepresentation3.EdgeColor = [0.0, 0.0, 0.50000762951094835]
DataRepresentation3.SelectionPointFieldDataArrayName = 'MagVort'
DataRepresentation3.SelectionCellFieldDataArrayName = 'Vorticity'
DataRepresentation3.ScalarOpacityFunction = a1_MagVort_PiecewiseFunction
DataRepresentation3.ColorArrayName = ('POINT_DATA', 'MagVort')
DataRepresentation3.ScalarOpacityUnitDistance = 0.00036236486270562699
DataRepresentation3.Visibility = 0
DataRepresentation3.LookupTable = a1_MagVort_PVLookupTable
DataRepresentation3.ExtractedBlockIndex = 1
DataRepresentation3.ScaleFactor = 0.0100000001490116

SetActiveSource(Contour2)
DataRepresentation4 = Show()
DataRepresentation4.EdgeColor = [0.0, 0.0, 0.50000762951094835]
DataRepresentation4.SelectionPointFieldDataArrayName = 'cell_velocity_pn'
DataRepresentation4.SelectionCellFieldDataArrayName = 'Vorticity'
DataRepresentation4.ColorArrayName = ('POINT_DATA', 'GasVort')
DataRepresentation4.Visibility = 0
DataRepresentation4.LookupTable = a3_GasVort_PVLookupTable
DataRepresentation4.ScaleFactor = 0.00095903314650058703

SetActiveSource(Reflect2)
DataRepresentation6 = Show()
DataRepresentation6.EdgeColor = [0.0, 0.0, 0.50000762951094835]
DataRepresentation6.SelectionPointFieldDataArrayName = 'cell_velocity_pn'
DataRepresentation6.SelectionCellFieldDataArrayName = 'Vorticity'
DataRepresentation6.ScalarOpacityFunction = a3_GasVort_PiecewiseFunction
DataRepresentation6.ColorArrayName = ('POINT_DATA', 'GasVort')
DataRepresentation6.ScalarOpacityUnitDistance = 0.00019599788738142
DataRepresentation6.LookupTable = a3_GasVort_PVLookupTable
DataRepresentation6.ExtractedBlockIndex = 1
DataRepresentation6.ScaleFactor = 0.00095903314650058703
```

Fig. 4(g). Listing (continued) of the Python pipeline file "VortTubes.py" used with Catalyst in RAGE to generate the time snapshots of the 3D RAGE simulation shown in Fig. 5.

```
      annotationColor = [1.0, 1.0, 1.0]
      VersionText = Text(Text='3D Rage Omega Capsule P30 Asymmetry 50%')
      VersionRep = Show()
      VersionRep.WindowLocation = 'UpperCenter'
      VersionRep.Color = annotationColor
      VersionRep.FontSize = 10
      VersionRep.Orientation = 0
      VersionRep.TextScaleMode = 'Viewport'

      AnnotateTimeFilter1 = AnnotateTimeFilter()
      TimeRep = Show()
      AnnotateTimeFilter1.Format = 'time = %6.4e s'
      TimeRep.WindowLocation = 'LowerLeftCorner'
      TimeRep.Color = annotationColor
      TimeRep.FontSize = 10
      TimeRep.TextScaleMode = 'Viewport'

    return Pipeline()

  class CoProcessor(coprocessing.CoProcessor):
    def CreatePipeline(self, datadescription):
      self.Pipeline = _CreatePipeline(self, datadescription)

  coprocessor = CoProcessor()
  freqs = {'input': [1]}
  coprocessor.SetUpdateFrequencies(freqs)
  return coprocessor

coprocessor = CreateCoProcessor()

# ---------------------- Data Selection method ----------------------

def RequestDataDescription(datadescription):
    "Callback to populate the request for current timestep"
    global coprocessor

    if datadescription.GetForceOutput() == True:
        for i in range(datadescription.GetNumberOfInputDescriptions()):
            datadescription.GetInputDescription(i).AllFieldsOn()
            datadescription.GetInputDescription(i).GenerateMeshOn()
        return

    for input_name in simulation_input_map.values():
       coprocessor.LoadRequestedData(datadescription)

# ------------------------ Processing method ------------------------

def DoCoProcessing(datadescription):
    "Callback to do co-processing for current timestep"
    global coprocessor

    timestep = datadescription.GetTimeStep()

    # Update the coprocessor by providing it the newly generated simulation data.
    # If the pipeline hasn't been setup yet, this will setup the pipeline.
    coprocessor.UpdateProducers(datadescription)
```

Fig. 4(h). Listing (continued) of the Python pipeline file "VortTubes.py" used with Catalyst in RAGE to generate the time snapshots of the 3D RAGE simulation shown in Fig. 5.

```
    # Write output data, if appropriate.
    #coprocessor.WriteData(datadescription);

    # Write image capture (Last arg: rescale lookup table), if appropriate.
    coprocessor.WriteImages(datadescription, rescale_lookuptable=False)

    # Live Visualization, if enabled.
    #coprocessor.DoLiveVisualization(datadescription, "localhost", 22222)
```

Fig. 4(i). Listing (continued) of the Python pipeline file "VortTubes.py" used with Catalyst in RAGE to generate the time snapshots of the 3D RAGE simulation shown in Fig. 5.

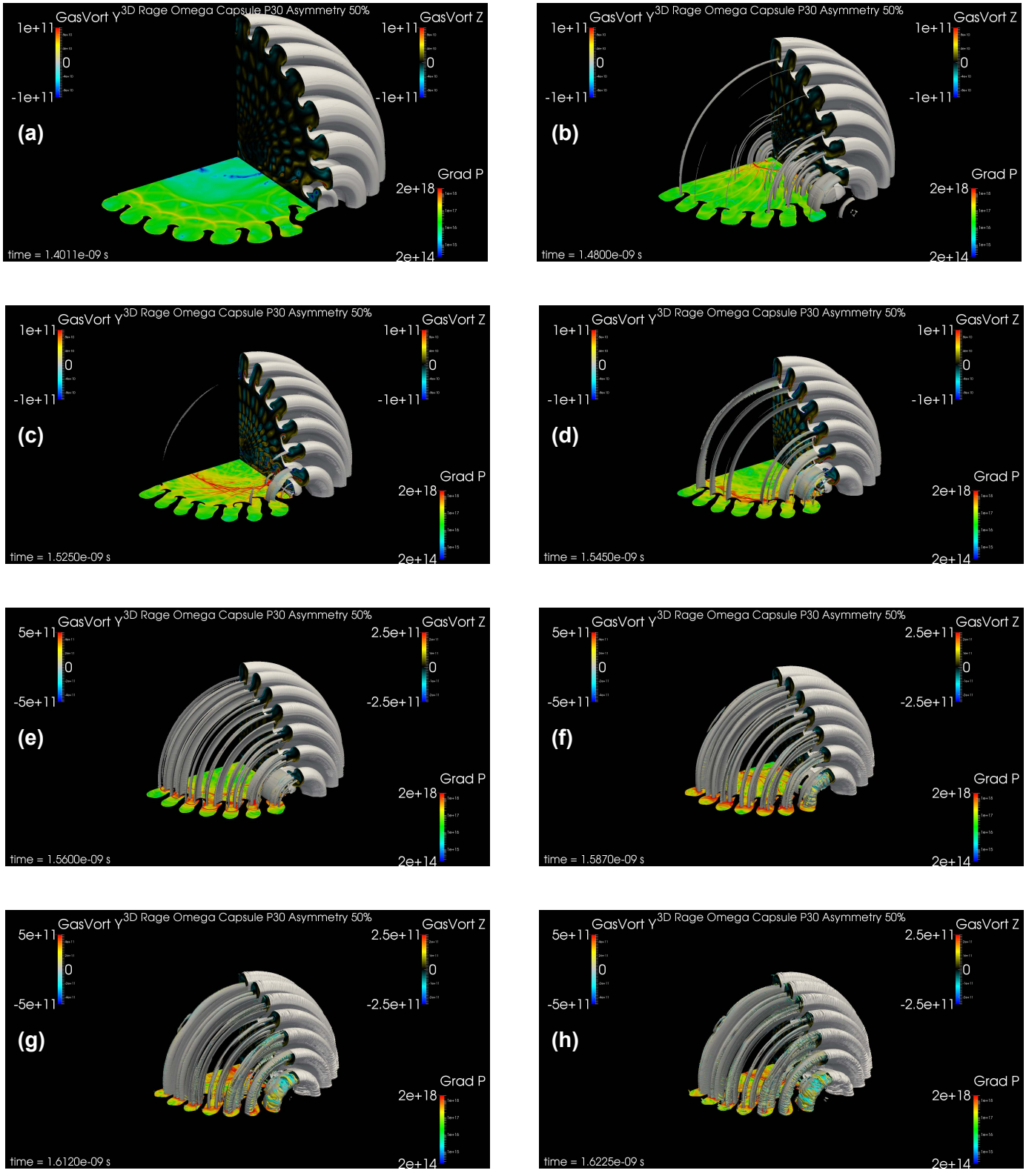

Fig. 5. Eight time snapshots from the 0.05 μm 3D RAGE simulation of the P30 OMEGA capsule generated using the in-situ visualization capabilities of RAGE. The gray vortex tubes with yellow and blue regions are isosurfaces of total vorticity at a constant value of $5\times10^{10}\ \mathrm{sec}^{-1}$ in (a) – (b) and $2.5\times10^{11}\ \mathrm{sec}^{-1}$ in (c) – (h).

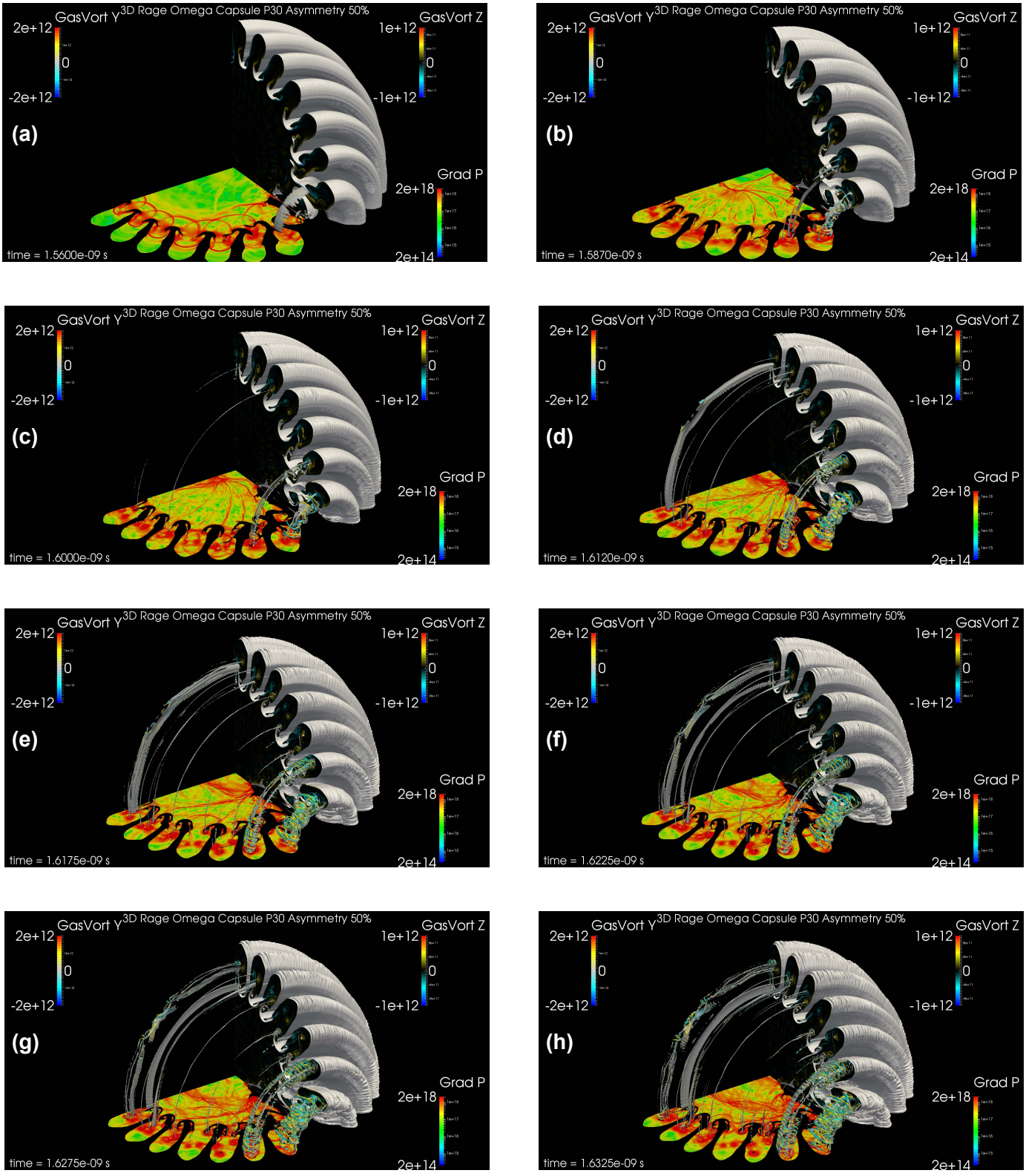

Fig. 6. Eight time snapshots from the 0.05 μm 3D RAGE simulation generated using the in-situ visualization capabilities of RAGE showing a close up view of the capsule. The gray vortex tubes with yellow and blue regions are isosurfaces of total vorticity at a constant value of $1\times10^{12}\,\mathrm{sec}^{-1}$.